SOFTWARE DEVELOPMENT FOR MULTI-LEVEL PETRI NET BASED DESIGN INFERENCE NETWORK

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

ÇAĞDAŞ COŞKUN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
MECHANICAL ENGINEERING

JULY 2004

Approval of the Graduate School of Natural and Applied Sciences

Prof. Dr. Canan Özgen
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Prof. Dr. Kemal İder
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Prof. Dr. Abdülkadir Erden
Supervisor

Examining Committee Members:

Prof. Dr. Bülent Platin Emre	METU, ME	
Prof. Dr. Abdülkadir Erden	METU, ME	
Asst. Prof. Dr. Zühal Erden	ATILIM UNV., IE	
Prof. Dr. Metin Akkök	METU, ME	
Prof. Dr. Sç Engin Kılıç	METU, ME	

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name : Çağdaş Coşkun

Signature :

ABSTRACT

SOFTWARE DEVELOPMENT FOR MULTI-LEVEL PETRI NET BASED DESIGN INFERENCE NETWORK

COŞKUN, Çağdaş
M.S., Department of Mechanical Engineering
Supervisor : Prof. Dr. Abdülkadir ERDEN

July 2004, 118 Pages

This thesis presents the computer implementation of a multi resolutional concurrent, design inference network, whose nodes are refined by PNDN (Petri Net Based Design Inference Network) modules. The extended design network is named as N-PNDN and consists of several embedded PNDN modules which models the information flow on a functional basis to facilitate the design automation at the conceptual design phase of an engineering design.

Information flow in N-PNDN occurs between parent and child PNDN modules in a hierarchical structure and is provided by the token flow between the modules. In this study, computer implementation of the design network construction and token flow algorithms for the N-PNDN structure is restored and therefore the previous DNS (Design Network Simulator) is adapted for the multi layer design and decomposition of mechatronic

products. The related algorithms are developed by using an object oriented, visual programming environment. The graphical user interface is also modified. The further developed DNS has been used for the application of the N-PNDN structure in the conceptual design of 5 mechatronic systems.

In the guidance of this study, it has been understood that the further developed DNS is a powerful tool for designers coming from different engineering disciplines in order to interchange their ideas.

Keywords: Design Network Simulator (DNS), Petri Net, Modularity, Functional Decomposition, Mechatronic Design

ÖZ

ÇOK KATMANLI PETRİ NET TABANLI TASARIM-ÇIKARIM AĞI İÇİN ALGORİTMA GELİŞTİRİLMESİ

COŞKUN, Çağdaş
M.S., Department of Mechanical Engineering
Supervisor : Prof. Dr.Abdülkadir ERDEN

Temmuz 2004, 118 Sayfa

Bu tezde düğümleri PNDN modülleriyle modellenmiş çok katmanlı, eş zamanlı bir tasarım çıkarım ağının bilgisayar uygulaması sunulmaktadır. Bu tasarım ağı PNDN modüllerinden oluşan ağ anlamında N-PNDN olarak adlandırılmıştır ve PNDN ağının içine gömülmüş pek çok PNDN modülünden oluşmakta olup, bir mühendislik tasarımında kavramsal tasarım otomasyonunu kolaylaştımak amacıyla bilgi akışı fonksiyonel temelde modellenmiştir.

N-PNDN'de bilgi akışı üst ve alt PNDN modülleri arasında gerçekleşmektedir, bu sebeple yapı hiyerarşiktir. Modüller arası bilgi akışı simge akışı ile gösterilmektedir. Bu tez çalışmasında çok katmanlı tasarım-çıkarım ağının kurulması ve bu ağdaki bilgi akışının bilgisayar uygulaması yeniden yapılandırılmıştır. Bu nedenle, bir önceki Petri net tabanlı tasarım

çıkarım ağı için geliştirilen algoritma, mekatronik ürünlerin çok katmanlı ve işlevsel tasarımına imkan verecek hale getirilmiştir. İlgili algoritmalar bir nesneye yönelik görsel programlama ortamı kullanılarak tamamlanmıştır. Buna ek olarak grafik kullanıcı arayüzü de son yapılan değişikliklere uygun hale getirilmiştir. Geliştirilmiş DNS, 5 mekatronik sistemin kavramsal tasarımında N-PNDN mimarisinin uygulaması için kullanılmıştır.

Bu çalışmaların sonucunda, DNS'in tasarımcılar arasındaki fikir alışverişini kolaylaştıran, görsel zeminde detaylı bir arayüz sunduğu gözlemlenmiştir.

Anahtar Kelimeler: Tasarım Ağı Benzetimcisi (DNS), Petri Net, Modülerlik, İşlevsel Tasarım, Mekatronik Tasarım

TO MY FAMILY

ACKNOWLEDGMENTS

I would like to express sincere appreciation to Prof. Dr. Abdülkadir Erden for his valuable guidance, encouragement, insight and patience throughout the study. I would also like to thank Dr. Zühal Erden for conducting a doctoral thesis on such an interesting and challenging topic, and for providing me the knowledge.

I am grateful to Serkan Güroğlu for his continuous help, encouragement, creative ideas and support.

Finally I would like to thank my mother and sister for their never ending love, support and patience in every stage of my life as well as throughout this study.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF SYMBOLS

CD	Compact Disc
df_i	Decision function
DNS	Design Network Simulator
FDT	Functional Design Tree
F_i	Functional State
FS	Functional State Set
GUI	Graphical User Interface
I	Input Mapping
M^I	Instantiation Marking
M^v	Variable Marking
PNDN	Petri Net Based Design Inference Network
O	Output Mapping

CHAPTER 1

INTRODUCTION

Automation of design process is being imposed as an imperative enrollment in engineering applications and this feature necessitates development of new tools in order to achieve a faster and more successful design process.

Design automation aims to minimize the human involvement at each step of design process by the help of systematic computer implementation where necessary. This automation provides guidance to the human designer and makes it easy to create various design alternatives. These alternatives can be handled faster leading to better designs with improved functionality, low cost and better quality which means a better and pretentious place in the competitive market.

If the flow of work in engineering design is considered, the phases are:

1. Clarification of the task – determines the definition of the problem, requirements to be fulfilled and the constraints.

2. Conceptual Design – involves client requirements and constraints, the evaluation of concept variants against design objectives, the definition of function structures and creation of design alternatives.

3. Embodiment Design – selects the proper standard elements in order to perform these functions together with the determination of layout and development of a technical product or system.

4. Detail Design and Documentation – is the phase where all production documents are prepared for the technical and economical calculations.

It can easily be seen that the conceptual design phase is the vital stage for the automation of design phase and it differs from traditional detailed design in the aspect of faster and efficient designs.

The conceptual design is used for evaluating the contents of a potential production project in an early phase, thus providing the customer with the best possible input for deciding on the future of the project and makes it possible to assess different solutions and thereby create a broad and sound foundation for making decisions with regard to time, solutions, and price. It also combines both engineering creativity and human intelligence. In addition to those advantages, the conceptual design phase provides an information transfer to the embodiment design, detail design and documentation phases. Therefore it plays the most important role in the automation and a special emphasis should be given.

The automation in engineering design is achieved to a certain extend in detail design and documentation, through the use of commercial software packages like MATHCAD MATLAB, ANSYS, etc...While those software programs have dramatically decreased the time required to move a new concept from design through manufacture and production, there are few tools that specialize in driving that process at the earliest phases of design. Engineers need innovative engineering tools to increase their productivity

during the conceptual design phase rather than the core detailed design through manufacturing process.

A functional and task independent design model was previously developed and applied to mechatronic design problems in order to accomplish this automation. This function based design architecture is called PNDN (Petri Net Based Design Network) (Erden, 1999). It is a design inference network that supports domain integration for the conceptual design of mechatronic products. It is based on the Petri Net theory (Reisig, 1985; Reisig, 1992).

Hybrid Automata (Alur et al., 1994) while considering those engineering systems are hybrid systems that consist of both discrete and continuous behaviors. It provides a mathematical model that combines the discrete dynamics of finite automation with the continuous dynamics of a dynamical system. While designing network model, the hybrid automata is used as an intermediate framework to support automatic transition from the functional representation of a system to the PNDN of the same system.

PNDN has following features:

1. Engineers from different engineering disciplines do not have time for training and must become productive rapidly without learning specialized skills. Therefore team members can easily use this tool.
2. It gives the ability to evaluate a number of functional design alternatives in a short time.
3. It supports the information flow.
4. Uncertainties can be handled and can be reduced during the information flow.

The information flow is represented by a token flow in PNDN and this token flow is applicable to both deterministic and non-deterministic PNDN.

PNDN theory provided an important progress in the automation of conceptual design phase. On the other hand, during the detail design automation and manufacturing automation, the relevant design phase needs a computer implementation. The computer implementation of PNDN is accomplished (DNS - Design Network Simulator) and has been developed in Borland C++ Builder integrated development environment (Güroğlu, 1999).

With the introduction of DNS (Design Network Simulator), the information flow and interchange of different design alternatives among team members, simulation functionality both for deterministic and non-deterministic token flow models, easy manipulation of previously created designs, have been accomplished.

PNDN can also handle the lower functional hierarchy levels of mechatronic products. A network of N-PNDN - defined as multi resolutional design inference network which is based on the modularity feature of PNDN and understanding of Function-Subfunction structure existing in the functional decomposition, (Korkmazel, 2001) - modules are used to model the product in every resolution level. In this model the structure of information flow between modules of PNDN fits into a hierarchical definition that makes the vertical communication between subordinate and super-ordinate modules possible. This is a good way of controlling the complexity of systems and therefore the information flow is utilized in N-PNDN.

This thesis focuses on computer implementation of N-PNDN theory which consists of design network creation of those N-PNDN modules.

The software will be the modification of DNS (Güroğlu, 1999) software for the N-PNDN theory and it should support the following features:

1. N-PNDN modules should be constructed separately and for every resolution level.
2. The N-PNDN modules should be independent of each other.
3. It should allow modeling the complex systems and therefore high resolution levels can easily be handled.
4. The information flow which is represented by token flow in each level and from one level to another should be provided.
5. The modified software can easily be used with minimum computer hardware resources.
6. The software should support the easy manipulation of previously created designs.
7. The software should give the possibility of further modifications other than information flow like material and energy flow at lower levels of resolution.

SCOPE OF THE RESEARCH

The main objective of the present research is to develop a new algorithm for the improved N-PNDN theory (Korkmazel, 2001) in order to implement the modular and multi level resolution of functional states in a computer environment.

In this thesis the algorithm and implementation of this algorithm for deterministic token flow simulation and analysis parts for N-PNDN modules are developed and case studies are carried out to evaluate and verify the range of applicability of the N-PNDN model.

While decomposing the functional model of mechatronic products, a vertical hierarchical design tree is considered. Therefore models can be thought as a collection of sub models and the process of hierarchically

decomposing of a model into a more detailed and complex sub models is called hierarchical refinement. If the refinement is accomplished by substituting models with the same functionality but with more detailed and accurate results then this refinement is called as a horizontal refinement.

One of the main criteria, in order to accomplish a complete conceptual design network using PNDN theory, is the vertical refinement for the modeling. Both in PNDN and N-PNDN theories this criterion is taken into account during functional modeling of the products.

While developing the algorithm for the N-PNDN, the modularity feature is also considered which aims to identify independent, standardized interchangeable units. Function modules help to implement technical functions independently or with other functions. Function modules are classified as basic, auxiliary, adaptive, and non modules according to (Pahl and Beitz, 1988).

The goals of research as follows:

1. Developing an algorithm for the construction of a multi-level PNDN structure.
2. Developing an algorithm for the token flow in the multi level resolution.
3. Developing an algorithm for the transition from one level to another without loosing the related data which belongs to the previous design or level.
4. Providing a user friendly GUI (Graphical User Interface) so that the people coming from different engineering disciplines can easily use the software package.

This thesis is composed of 6 chapters. Chapter 2 deals with the literature survey of previous studies about PNDN and N-PNDN. Chapter 3

concentrates on the N-PNDN theory and the methodology of representation of mechatronic products and their computer implementations. Chapter 4 involves the detailed understanding and explanation of further developed DNS software for the improved PNDN theory. The algorithms that are developed for the N-PNDN modules are presented. Chapter 5 includes the case studies and explanations. Finally, discussions, conclusions and recommendations for future work are provided in Chapter 6.

CHAPTER 2

LITERATURE SURVEY

This thesis focuses on the computer implementation of a multi-resolution functional model of mechatronic products by N-PNDN structure. Since N-PNDN is an inheritance of PNDN (Erden, 1999) the structure, literature survey of design concepts, previous algorithms of conceptual mechatronic design and Petri Nets will be handled respectively.

2.1 LITERATURE SURVEY ON THE CONCEPTUAL MECHATRONIC DESIGN

The involvement of faster microprocessors and developments in computer technology contributes the evolution of a new concept called mechatronics. In order to perform required functions, this new discipline involves the synergistic integration of software based control systems, electronic devices and mechanical structures in the same product (Fraser and Milne, 1994; Buur, 1992; Acar, 1993). At the same time, this integration plays a crucial role for further development of mechanical elements, machines, and precision mechanics toward mechatronic systems (R. Isermann, 1996).

According to (Amerongen, 2003), design of a mechatronic system covers design of the mechanical structure and its embedded control system. Therefore, previously developed design strategies for other engineering disciplines, are not applicable for the development of mechatronic products.

Pahl and Beitz (Pahl and Beitz, 1988), who had performed one of the important previous efforts to systematize design process of products, give the generally accepted flow of work definition throughout the design as follows:

- Clarification of the task,
- Conceptual design,
- Embodiment design,
- Detail design

Conceptual design phase, which determines the working principles of a product, is the major concern of this literature survey.

During conceptual design, the aim is not to complete a final design, but rather to identify the performance-limiting factors of the design proposal(s) and to choose satisfactory specifications for these factors (Coelingh, 2002).

According to (Rzevski, 2003), conceptual design is an early stage of design in which designers select concepts that will be employed in solving a given design problem and decide how to interconnect these concepts into an appropriate system architecture. For theoretical functional modeling to be applied early in design, the work-flow must always go from global or abstract to detailed or particular design specifications (Aleixos, Company, Contero, 2004). This avoids undertaking different design alternatives by reducing the uncertainty in a controlled manner and delays to have a particular design solution at the beginning of the design process.

Another definition of the conceptual design phase states that it is an early stage of the product development process having characteristics of fuzzy

problems, tolerating a high degree of uncertainty (Quin, Harrison, West, Jordanov, Wright, 2003).

As an example for design automation tool, Schemebuilder Mechatronics (Porter, Council, 1998), is an intelligent knowledge-based computer software for the conceptual design of mechatronic systems. The software is the combination of expert knowledge (rules) with object oriented representation together with simulation module where the expert knowledge depends on design principles that consist of physical understanding, design rules and observations. The knowledge base enables to resuse of previously generated design information and to infer new design rules by interacting with the user (Porter, 2002). In software package the modeling of designs (schemes) is achieved by creating models from object oriented descriptions of conceptual models. The objects which represents the real physical objects are connected with Bond Graph type ports. Schemebuilder represents designs as schemes based on the Function/Means Tree approach. In this approach functions are represented as the leaves of the tree. However, the function definition of this study is not flexible and abstract enough. Therefore it restricts the designer and prevents to think independently.

The system theory (Boardman, 1990) explains the basics of the mechatronics by a set of rules for abstract modeling of technical artifacts and hierarchically decomposed form of those artifacts as subsystems. A decomposition hierarchy places a partial ordering on the systems and their interrelationships such that higher level systems and their relationships are compositions of lower level systems and relationships (Kannapan and Marshek, 1991). Decomposition approach reduces the complexity of the system by dividing it into sub modules which means easy management and implementation.

Hierarchical decomposition approach is applied at the two levels of mechatronic systems, yielding “Functional Decomposition” and “Structural Decomposition” concepts be developed (Korkmazel, 2001).

Functional decomposition is defined as partitioning a given complex functional structure hierarchically into more manageable functions such that it is easier to match design concepts with these functions and arrive at a solution to the problem (Korkmazel, 2001). At the same time, structural decomposition approach is used in redesign procedure or in reverse engineering problems which leads to restrict the solution and design process in a small sub module of the system.

The literature search has revealed that automation of conceptual design phase of mechatronic products necessitates a functional approach to the problem in order to develop a systematic design process. Considering the importance of conceptual design which generates design alternatives, design automation requires system functions to accomplish for given task. Therefore different functional decomposition approaches besides functional design tree are used for to define functional structure of mechatronic systems.

From the mathematical point of view, functions can be divided into two as continuous and discontinuous. The mechatronic system is composed of both continuous and discontinuous behaviors (Erden, 1999). For this reason an integrated view, which comprises from these behaviors, is required. In order to satisfy this need PNDN (Petri Net Based Design Inference Network) has been developed by Erden (Erden et. al., 1999).

2.2 FUNCTIONAL REPRESENTATION SCHEMES

In order to create the functional structure of a system, different representation schemes are used. These are Functional Block Diagrams, Functional Decomposition Tree, Function/Means Trees, FR/DP (Functional Requirement Design Parameters) and AND/OR Trees.

2.2.1 Functional Block Diagram

According to (Pahl and Beitz, 1988) and (Blanchard and Fabrycky, 1998) functional block diagram is mentioned as a convenient mechanism for communicating the functional information of an artifact. They structure the system requirements and represent them as functions in a sequence together with their series and parallel interrelationships. Since the sequences of functions are not primarily considered in PNDN approach, the FBD (Functional Block Diagram) methodology is not appropriate for this study.

2.2.2 AND/OR Tree

AND/OR tree defines design requirements and design specifications in the form of a hierarchy (Kusiak et al., 1991). A desired requirement is divided into sub requirements until it matches its corresponding function. Unfortunately this methodology needs the rules of passing from functional domain to physical domain; therefore it is not suitable for this study.

2.2.3 Functional Design Tree

The functional structure of a system is presented hierarchically in "Functional Design Tree (FDT)" with the definitions of "Functional Cells (FC)" and "Atomic Functional Cells (AFC)" (Erden, 1996; Erden, 1999). In this hierarchic structure "Functional Cells" and "Atomic Functional Cells"

correspond to subfunctions of the system at different levels and leaves of the tree respectively. Towards the leaves of the tree, in other words from more abstract representation to a particular solution, FCs gain precision and AFCs find their interpretation as a component of a machine element or a formula based representation. However FDT lacks of means for each function which is necessary for further decomposition and embodiment design.

2.2.4 FR/DP Tree

According to (Suh, 1998), the system is defined as the assemblage of sub-systems, hardware and software components, and people designed to perform a set of tasks to satisfy specified functional requirements and constraints. The first step in design process of a system is determining the customer needs and the functional requirements (a minimum set of independent requirements that completely characterizes the functional needs of the product (or software, organizations, systems, etc.) in the functional domain) together with constraints of the system (Suh, 1998). The functional requirements should be specified without thinking the solution in the sense of creativity and close to the customer needs. Next step is the conceptualization process which is to map the FRs of the functional domain into physical domain by identifying the design parameters (DP) that characterize the design. Design Parameters can be a mechanical component, a sensor or a computer code depending on the design. Having chosen the design parameters, process variables (PV) should be identified that generate the specified design parameters.

The mapping should support the independence axiom which quarentees the independence of functional requirements. An ideal design is the design that has equal FRs and DPs. According to the information axiom the design with least information is the best design. The information axiom guides the designer to select the DPs and helps the designer to select the best design

among others. It is the best tool when there are more than one functional solution.

2.2.5 Function/Means Tree

The Function/Means tree is a graphical representation (Andreasen, 1980) based on the Hubka's "Law of Vertical Causality" (Hubka and Eder, 1988). The "Law of Vertical Causality" states that the decomposition of a particular function into subfunctions is only possible, when a means has been chosen to realize the function (Andreasen, 1980). Means can be a particular solution to a problem, an organ, a machine part, a detailed component which can realize the function. Number of alternative means can be proposed once the function is identified. According to the "Law of Vertical Causality", there should be a causal relationship between the functions and means that realize them. As the decomposition proceeds, the means becomes simpler and that results in less complex modules. Function/Means tree is a practical tool for analysis work and one can visualize the design alternatives in more effective way. It supports the designer in thinking about his/her design in an axiomatic way (Suh, 1998), which enables some quick feedback concerning the completeness and quality (clarity) of a design (Ringstad, 1997).

2.3 BEHAVIORAL DESIGN

As a definition behavior is the response of an artifact to its environment. In the same manner, behavioral design aims to define states and state change conditions of the considered design. The following sections mention some behavioral design techniques in literature namely, Finite Automata, Hybrid Automata, Discrete Event System Modeling and Petri Nets.

2.3.1 Finite Automata

A finite automaton is defined as an abstract model describing a synchronous sequential machine (Kohavi, 1978; Hopcroft and Ullman, 1979; Lewis and Papadimitriu, 1981) where network outputs at any given time are functions of the external inputs and stored information at that time. The continuous behavior of the machine is not modeled and is described as a sequence of discrete events instead of continuous state changes (Erden, 1999).

2.3.2 Hybrid Automata

A hybrid automaton is a formal model for a hybrid system which can be described as a dynamic system with discrete and continuous components. (Henziger et al., 1995; Puri and Varaiya, 1994; Puri and Varaiya, 1995).

2.3.3 Discrete Event System Modeling

A discrete event system (DES) is a dynamic system that evolves in accordance with the abrupt occurrence, at possibly unknown irregular intervals, of physical events (Ramadage and Wonham, 1989; Mortazavian and Lin, 1991; Zeigler, 1989).

2.3.4 Petri Nets

Petri Nets are models for procedures, organizations and devices where regulated flows, in particular information flows play an important role (Reisig, 1985; Reisig, 1992; Tabak and Levis, 1985). As another definition Petri Nets are bipartite directed multi-graphs, which are used to model procedures, organizations and devices (systems in general), in which regulated flow of objects and information occurs (Andreadakis, 1988; Reisig, 1985; Reisig, 1992).

Petri Nets are under development since Carl Adam Petri has firstly defined language in 60's. It is the first system that discrete parallel system is defined and it is a generalization of automata theory such that the concept of concurrently occurring events can be expressed. By this way it is possible to model the dynamic behavior of the systems. Modeling, Control and Performance Analysis, Intelligent Task Planning, Management of Manufacturing Systems are the most common application areas of Petri Nets.

Petri Nets consist of passive components, active components and directed links. Passive components are the places which are denoted by circles (O). The active components are the transitions and denoted by boxes ( ). Directed links represent the abstract relationships between components.

2.4 Previous Petri Net Tools

Some of the previous Petri tools and their properties are given below. Additional and detailed information about Petri Tools can be found in Appendix.

2.4.1 Petri Tool

Petri Tool (Brink, 1996) was written in Java and it is one of the few tools which was developed in Java. Java is an Object Oriented language and at the same time it is an interpreted language. Therefore any machine with Java Virtual Machine can run this software providing a high portability feature.

2.4.2 Cabarnet

Cabarnet (Computer Aided software engineering environment Based on ERNETs) was developed for the real time systems. Robot Arm control was one of its strong application area. This tool was written in C++ in UNIX environment as operating system.

2.4.3 Alpha/Sim

ALPHA/Sim is a general purpose graphical discrete-event simulation tool based on Petri Nets. Communication networks and computer, flexible manufacturing systems are the basic application areas (AlphaTech, 2004).

2.4.4 PNDN

PNDN is a Petri Net-based Design Network which was developed by (Erden, 1999) and developed for the representation and analysis of the functions and their interrelationships through information flow for the conceptual design stage of engineering systems and at the first level of design. PNDN structure accomplished a great improvement in the automation of conceptual design phase. The computer implementation of PNDN was accomplished for the first decomposition level in Borland C++ Builder IDE (Integrated Development Environment) which was called DNS (Güroğlu, 1999).

2.4.5 N-PNDN

N-PNDN is the extension of PNDN (Korkmazel, 2001) which is an improved model and obtained multi-resolutional feature in terms of functionality and modeling. Functional decomposition rules were utilized in the extended design inference network, N-PNDN, were provided. Details of

computer implementation of N-PNDN and the contributions of this study will be given in the next chapters.

2.5 Evaluation of Literature Survey

In the light of the facts mentioned in previous sections, Function/Means Trees are more useful for representing different design alternatives during the conceptual design phase. Function/Means tree can model and represent the whole design process from the most abstract level representation to the specific descriptions of the solutions to problems. It enables the designer to evaluate different design alternatives by providing various kinds of means for each subfunction. Therefore, designer's ability to select functions and subfunctions independently results in high engineering creativity. Passing from functional domain to physical domain is easier than the other functional approaches.

The above mentioned Petri Net tools support only Petri Nets which is based on Place-Transition Nets or time dependent Petri Nets. On the other hand PNDN reveals a new approach to design modeling and defines new network elements. This entails to develop a new design modeling tool for the mechatronic products. The survey on Petri Net tools showed that the new developed tool should not need additional programming effort. Moreover, Windows or a X-Windows based operating system is much more convenient to use and these operating system environments present a sophisticated graphical user interface which one can not find at other operating systems. In addition to those major facilities of the software package such as printing, saving, griding properties should be developed. The survey showed that the recent software packages lack of these properties. Finally token flow feature should be provided in the new design tool in order to model the dynamic behavior of a system in multi level representation since multi level modeling is also a missing part of the current Petri Net software packages.

CHAPTER 3

BASICS OF N-PNDN

PNDN is a design inference network developed for the representation and analysis of the function and their interrelationship through information flow for the conceptual design stage of engineering systems in order to facilitate design automation. (Erden et al., 1998). The formal structure is based on the theory of Petri Nets and Hybrid Automata. Therefore the extended design network is named as the network of PNDN modules, N-PNDN, and consists of embedded PNDN modules in the topmost level PNDN network.

3.1 ARCHITECTURE OF N-PNDN

The N-PNDN theory handles the functional decomposition phase of the mechatronic products in multi resolutional level. For this reason it will be more convenient to give a design application of a mechatronic product in order to go to deep of the design details. Dish machine is taken as the sample mechatronic device to illustrate the procedure using N-PNDN in multi resolution levels.

Creation of N-PNDN consists of following steps:

- Functional Representation Using Functional Design Tree.
- Definition of Variables
- Definition of Instantiation of Variables
- Definition of Decision of Functions
- Definition of Input Mappings

- Definition of Output Mappings
- Repeat The Steps Given Above For Each Resolution Level of Each Functional State.
- Create and STOP Functional State for the each sub level of Functional State in order to finish the performance of the relevant function during token flow.

3.1.1 DEFINITION OF FUNCTIONAL STATE SET

The first step for the functional representation of a system is to establish a functional design tree, a hierarchical structure that involves sub-functions of systems at various levels of resolution where top most nod is to satisfy the required function.

$F(S) = \{F_1 , F_2 , \ldots\ldots\ldots\ldots F_N \}$ where ;

F(S) = Overall function of the required system.

F_i = Sub-function of the system at the first level of functional decomposition

N = Number of sub-functions

The functional states in Petri Nets are represented by transitions and denoted by (T). As mentioned before, the dish machine will be designed by N-PNDN and will consist of different layers. Therefore each layer of resolution involves different functional states. For the first layer and the sub layer the functional states are follows:

FIRST LEVEL:

$F (S) = \{ F_1 , F_2 , F_3 , F_4 \}$

F1 = START (CLEAN DISHES)

F2 = LOAD – UNLOAD DISHES

F3 = WASH DISHES

F4 = DRY

F5 = STOP

SECOND LEVEL

The functional state Wash Dishes can be decomposed to the following functional states:

F1= WASH DISHES
F2 = TAKE WATER IN
F3 = HEAT WATER
F4 = SUPPLY WATER TO PROPOLLER
F5 = ROTATE PROPOLLER
F6 = TAKE DETERGENT
F7 = TAKE WATER-OUT
F8 = STOP

THIRD LEVEL

The functional state “TAKE WATER IN” can also be decomposed into one more layer and the functional states follows

F1 = TAKE WATER IN
F2 = OPEN THE INLET VALVE
F3 = STOP

The functional design tree of the dish machine is given in the following page.

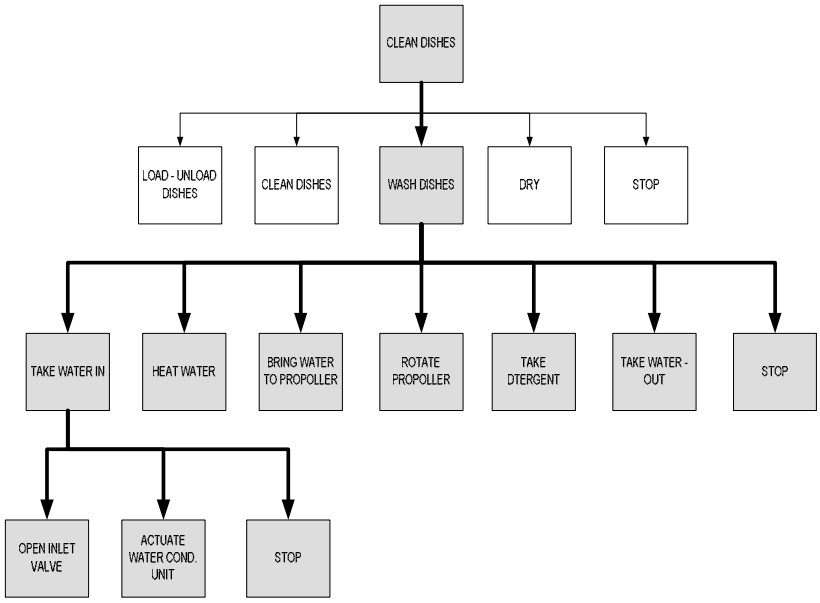


Figure 3.1 The Functional Design Tree of Dish Machine (Korkmazel, 2001)

3.1.2 DEFINITION OF VARIABLES

Variable sets are grouped into 2:

- Continuous Variable Set (CVS): represents the cont. behavior of the system.
- Discrete Variable Set (DVS): represents the type of configuration based information that is required to be perceived and controlled by the system to be designed.

In PNDN variables are defines as finite set of places and they are represented by the symbol (O)
Based on this information the variables have been given below for the each level of functional decomposition.

FIRST LEVEL

p_{10} = button (information about the start button, on or off)
p_{20} = clean (information about the cleanliness of the dishes)
p_{30} = door (information about the door of the machine, close or open

SECOND LEVEL

p_{10} = detergent (information for the amount of detergent)
p_{20} = water – temp (information for the water temperature)
p_{30} = water level (information for the water level)

THIRD LEVEL

p_{10} = water condition
p_{20} = filter

3.1.3 DEFINITION OF INSTANTIATIONS

Instantiations of the continuous variables are obtained by defining a threshold value to the continuous variables. They can take 1 or 0 in deterministic case and a value which varies between 0 and 1 for the non-deterministic case. They are also represented by the symbol (O) In dish machine design case:

FIRST LEVEL

p_{11} = on (button pressed)
p_{12} = off (button not pressed)
p_{21} = water – temp (dishes are clean)
p_{22} = water level (dishes are dirty)
p_{31}=closed (door is closed)
p_{32} = open (door is open)

SECOND LEVEL

p_{11} = det (there is detergent in the machine)
p_{12} = nodet (there is no detergent in the machine)
p_{21} = T>Ts (water temperature is higher than washing temperature)
p_{22} = T<Ts (water temperature is lower than washing temperature)
p_{23} = T=Ts (water temperature is equal to washing temperature)
p_{31} = excess (there is excess water in the machine)
p_{32} = enough (there is enough water in the machine)
p_{33} = not enough (there is not enough water in the machine)
p_{34} = none (there is no water in the machine)

THIRD LEVEL

p_{11} = good
p_{12} = nogood
p_{21} = notfull
p_{22} = full

3.1.4 DEFINITION OF DECISION FUNCTIONS

In PNDN, decision functions are represented as switches, which are the special form of transitions (Tabak and Levis, 1985). They are used in order to decide which instantiation is going to be fired and to process the corresponding variable.

In the dish machine example:

FIRST LEVEL

DF={df_1 , df_2 , df_3 }

df_1 = Decision Function for button variable
df_2 = Decision Function for clean variable
df_3 = Decision Function for door variable

SECOND LEVEL

DF={df_1 , df_2 , df_3 }

df_1 = Decision Function for detergent variable
df_2 = Decision Function for water-temp variable
df_3 = Decision Function for water level variable

THIRD LEVEL

DF={df_1 , df_2 }

df_1 = Decision Function for water-cond variable
df_2 = Decision Function for filter variable

3.1.5 DEFINITION OF I – MAPPINGS

By definition it is the mapping from places to transitions and represented by 0 and 1. There are 2 types of I-Mappings.

- Type 1 : I-Mapping from variables to its decision functions
- Type 2 : I-Mapping from instantiations to Functional states

The I-Mapping for the top and sub levels are given below:

FIRST LEVEL

Type 1

I (button, df_1) = 1 I(clean, df_2) =1 I (door, df_3) = 1

Type2

I (on, F_3) = 1 I (off, F_2) = 1 I (closed, F_3) = 1

I (on, F_4) = 1 I (clean, F_4) = 1 I (closed, F_4) = 1

I (on, F_5) = 1 I (dirty, F_3) = 1 I (open, F_2) = 1

I (open, F_5)=1

SECOND LEVEL

Type1

I (detergent, df_1) = 1 I(water temp, df_2) =1 I (water level, df_3) = 1

Type 2

I (det, F_6) = 1 I (nodet, F_7) = 1 I (none, F_8) = 1

I (T<Ts, F_3) = 1 I (nodet, F_8) = 1 I (nenough, F_4) = 1

I (T>Ts, F_2) = 1 I (nenough, F_2) = 1 I (enough, F_2) = 1

THIRD LEVEL

Type 1

I (water cond, df_1) = 1 I(filter, df_2) =1 I (good, F_2) = 1

Type 2

I (nogood, F_2) = 1	I (nogood, F_3) = 1	I (notfull, F_2) = 1
I (notfull, F_3) = 1	I (full, F_8) = 1	

3.1.6 DEFINITION OF O-MAPPINGS

By definition it is the mapping from transitions to places and represented by 0 and 1. There are 2 types of O-Mappings.

- Type 1 - O-Mapping from decision functions to instantiations
- Type 2 - O-Mapping of functional states

FIRST LEVEL

Type 1

O (df_1,on) = 1	O(df_2, clean) =1	O (df_3, closed) = 1
O (df_1,off) = 1	O(df_2, dirty) =1	O (df_3, open) = 1

Type 2

O(F_1, button) = 1	O(F_2, button) = 1	O(F_3, button) = 1
O(F_1, clean) = 1	O(F_2, clean) = 1	O(F_3, clean) = 1
O(F_1, door) = 1	O(F_2, door) = 1	O(F_3, door) = 1
O(F_4, button) = 1	O(F_4, clean) = 1	O(F_4, door) = 1

SECOND LEVEL

Type 1

O (df_1,det) = 1	O(df_2, T<Ts) =1	O (df_3, enough) = 1
O (df_1,nodet) = 1	O(df_2, T=Ts) =1	O (df_3, nenough) = 1
O (df_2,T>Ts) = 1	O(df_3, excess) =1	O (df_3, none) = 1

Type 2

O(F_1, detergent) = 1	O(F_1, water temp) = 1	O(F_1, water level) = 1
O(F_2, detergent) = 1	O(F_2, water temp) = 1	O(F_2, water level) = 1
O(F_3, detergent) = 1	O(F_3, water temp) = 1	O(F_3, water level) = 1
O(F_4, detergent) = 1	O(F_4, water temp) = 1	O(F_4, water level) = 1
O(F_5, detergent) = 1	O(F_5, water temp) = 1	O(F_5, water level) = 1
O(F_6, detergent) = 1	O(F_6, water temp) = 1	O(F_6, water level) = 1
O(F_7, detergent) = 1	O(F_7, water temp) = 1	O(F_7, water level) = 1

THIRD LEVEL

Type 1

O (df_1,good) = 1	O(df_2, nogood) =1	O (df_3, notfull) = 1
O (df_1, full) = 1		

Type 2

O(F_1, water cond) = 1	O(F_2, water cond) = 1	O(F_3, water cond) = 1
O(F_1, filter) = 1	O(F_2, filter) = 1	O(F_3, filter) = 1

Having defined the O-Mapping, the N-PNDN construction is completed for the dish machine design. As a result the NPNDN for the dish machine is given in Figure 3.2, Figure 3.3 and Figure 3.4

At the first level of clean dishes functional state decomposed into “Load-Unload”, “Wash Dishes”, “Dry”, “Stop” functional states. The variables “Button”, “Clean”, “Door” and the relevant instantiations have been connected through decision functions. At the second level “Wash Dishes” functional state decomposed into “Take Water In”, “Heat Water”, “Bring Water to Propeller”, “Rotate Propeller”, “Take Detergent”, “Take Water Out” and “Stop” functional states. Finally at the third level “Take Water In” functional state decomposed to “Open Inlet Valve”, “Actuate Water Condition Unit” and “Stop” functional states together with their variables which are denoted by place sets and instantiations.

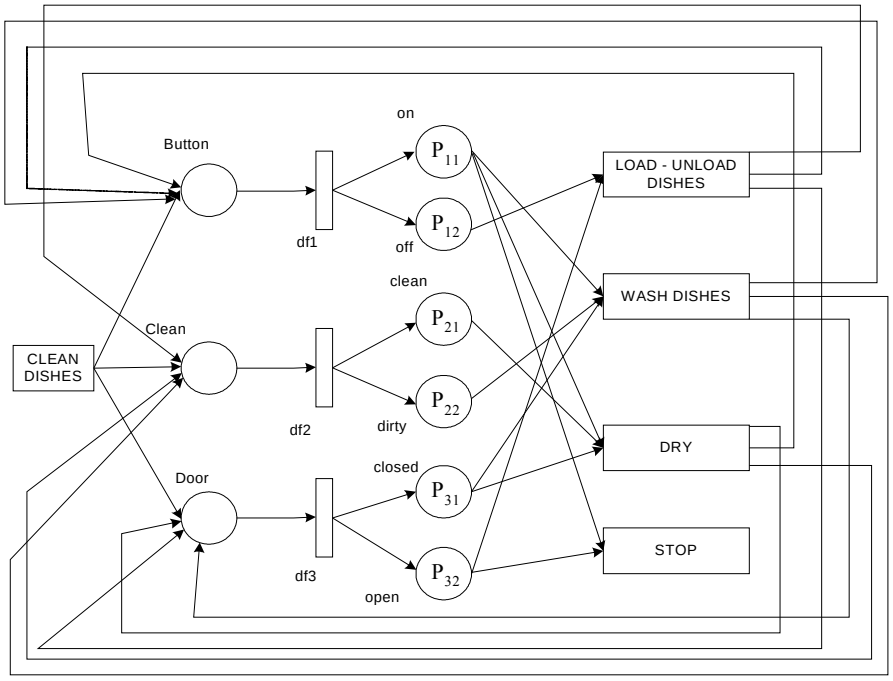


Figure 3.2 N-PNDN of the Dish Machine, the top most level of Dish Machine, (Korkmazel, 2001)

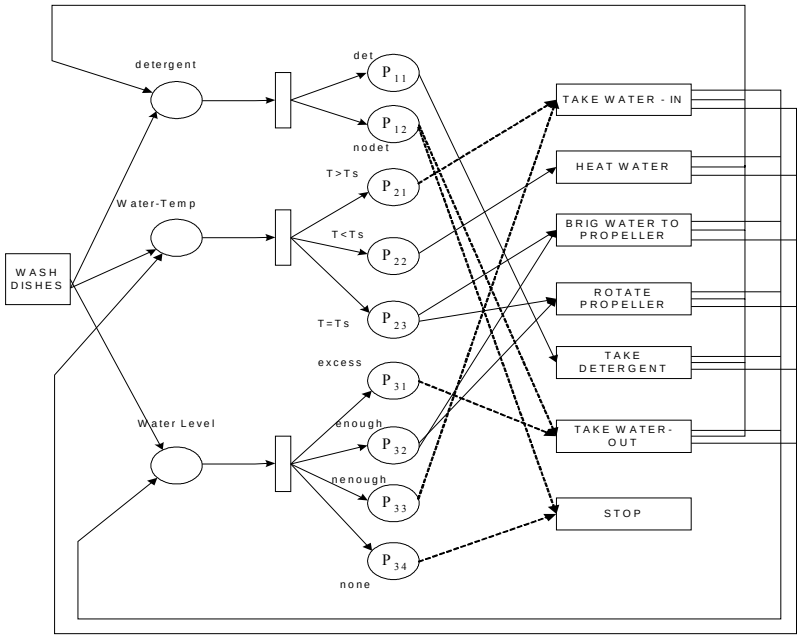


Figure 3.3 The sub-layer of the functional state Wash Dishes and its functional decomposition, (Korkmazel, 2001)

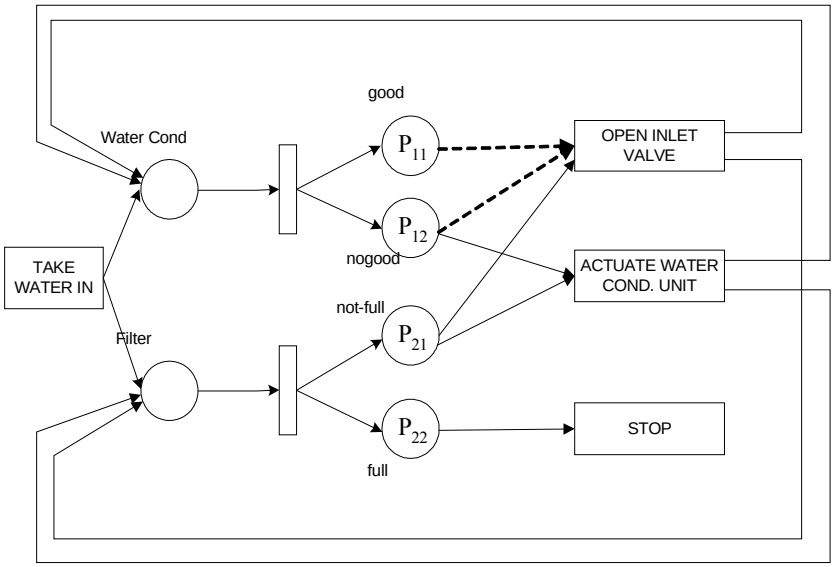


Figure 3.4 The sub-layer of the functional state Take Water - In and its functional decomposition, (Korkmazel, 2001)

3.2 TOKEN FLOW IN N-PNDN

The token flow in N-PNDN is the same with the PNDN; however token flow in each level is independent from other levels. Therefore the token flows are uncorrelated from each other. In order to understand the details of token flow in N-PNDN, some basic definitions should be explained.

In order to **enable** a decision function, it's input places should have token and all of its output places should have a token. This is called variable marking (M^V). If those conditions prevail, the enabled decision functions fire simultaneously by removing the input tokens from their places and then putting it in one of its transitions. This is called instantiation marking (M^I). Instantiation marking makes the transitions enable which means that all the instantiations (places) of the relevant transition have tokens on them. When a transition fires, it removes the token on its transitions and inserts it on its relevant places. This results in also variable marking (M^V). In N-PNDN structure if one of the functional state or in other words if one of the enabled transition has sub layer, the token goes through the start transition of the sub – layer and the token flow on the upper layer stops for a while until the token flow at the sub-level finishes. The termination criterion in order to finish a token flow is the visit of the token to the STOP transition. This means that the relevant function perform has finished and it is time to back to the upper layer. This goes on and repeats until all token flow at sub levels have finished, including every resolution level. Later on the token flow goes on like the single layer PNDN structure and the program restarts. Having first inserted and started the token flow from CLEAN DISHES transition, variable marking for dish machine is given starting from Figure 3.5 to Figure 3.12 step by step. Figure 3.5 shows variable (M^V) marking for dish machine at the top most level. Figure 3.6 gives Token Flow through decision functions to instantiations of variable places. In Figure 3.7 "WASH DISHES" transition fires and the token flows through the sub level of that

transition which is given in Figure 3.8. The token will come back to the top most level when it visits the stop transitions of its sub levels. When the WASH DISHES transition is fired, the variable marking automatically takes place and the token flows to the variable places of child PNDN of the WASH DISHES functional state. The variable marking is given in Figure 3.8 for the second level. The token flows through the decision functions to instantiation of variable places and is given in Figure 3.9. In Figure 3.10 "Take Water In" transition is fired (Korkmazel, 2001).

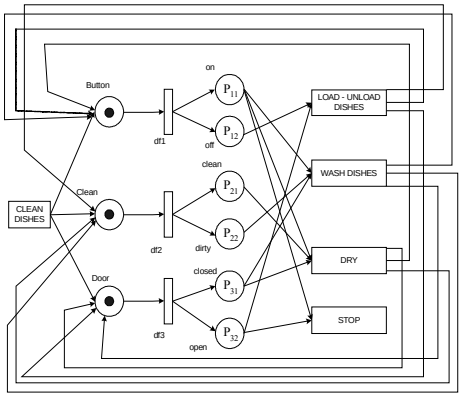


Figure 3.5 Token Flow First Step for Dish Machine (Korkmazel, 2001)

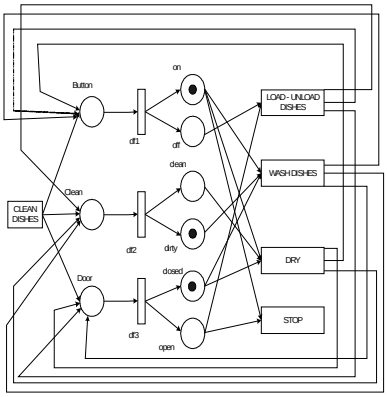


Figure 3.6 Token Flow Second Step for Dish Machine (Korkmazel, 2001)

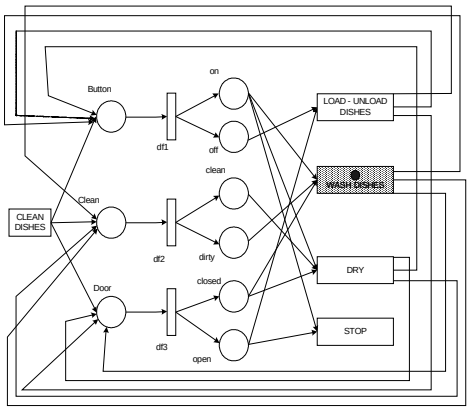


Figure 3.7 Token Flow Third Step for Dish Machine (Korkmazel, 2001)

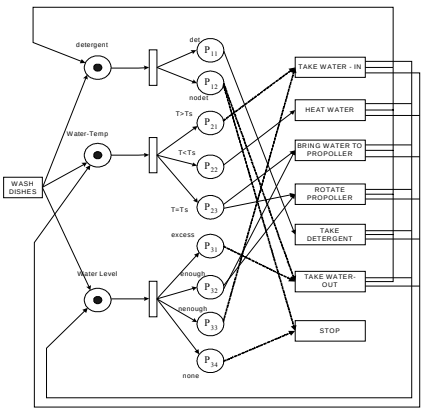


Figure 3.8 Token Flow Fourth Step for Dish Machine (Korkmazel, 2001)

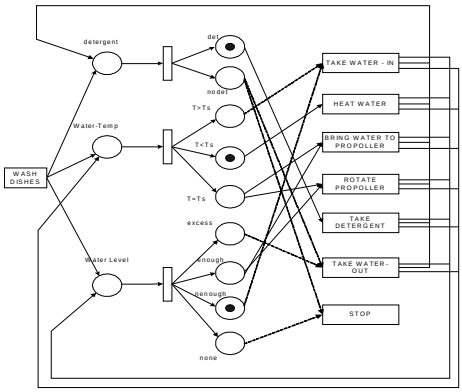


Figure 3.9 Token Flow
Fifth Step for Dish Machine
(Korkmazel, 2001)

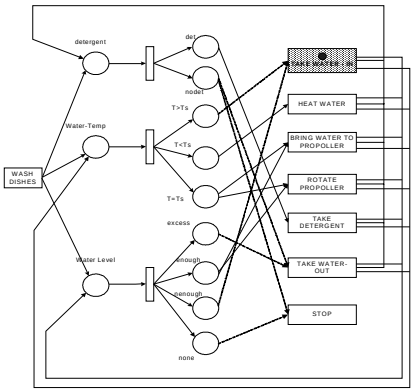


Figure 3.10 Token Flow
Sixth Step for Dish Machine
(Korkmazel, 2001)

Since the "Take Water In" transition has also a sub level the token will go through the sub level of the fired transition and the token flow starts at the final resolution level by variable marking (M^V) which is shown in Figure 3.11. Later on as shown in Figure 3.12 instantiation marking. The token flow will continue here unless the token visits the STOP transition. This is shown in Figure 3.13. When this happens the considered task for the functional state TAKE WATER – IN finishes and the token returns back again to its upper layer where first the TAKE WATER – IN transition has been fired and token flow restarts as shown in Figure 3.14.

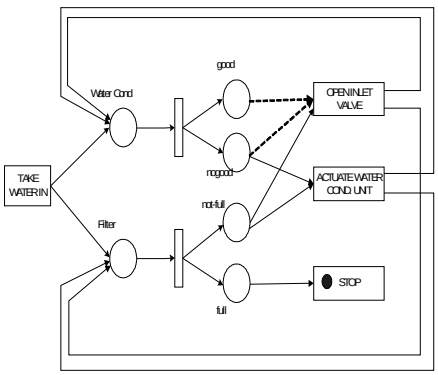


Figure 3.11 Token Flow
Seventh Step for Dish Machine
(Korkmazel, 2001)

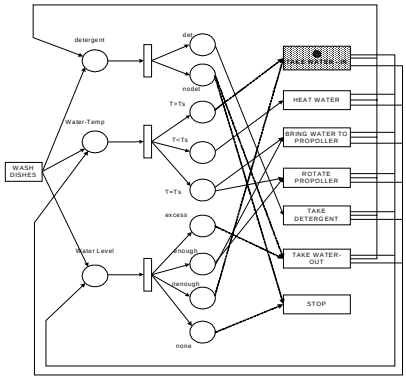


Figure 3.12 Token Flow
Eight Step for Dish Machine
(Korkmazel, 2001)

3.3 Evaluation of features of PNDN

In this section it has been shown that PNDN is a module which can be used to model every resolution level in the functional decomposition of mechatronic products. This modularity concept is used and as an application of this feature dish machine example is given. The term “Modularity” is used to describe the use of common units to create different products and it represents a common modeling unit for the different subfunctions of the system. Modularity is a powerful way of representing the flow of information contained within every subfunction of the system. Hence the subfunctions of the system are uncoupled which constitutes the functional independence. Using the common modeling approach following benefits has been provided:

Incrementability : The current functional model does not need any modification when it is necessary to add a new subfunction.

Modifiability: Each subfunction model is independent of each other so new replacements can be made independently.

Transparency: The whole subfunction structure makes the system understandable and the user can analyze the internal structure of subfunction modules easily.

Those modularity features of PNDN and therefore N-PNDN are explored by information flow between levels in dish machine example. For the modularity of PNDN, the “START” transition is replaced with another transition named after the subfunction it represents. This is the modification which is made in PNDN theory and that establishes the N-PNDN.

Mechatronic systems consist of sensory, cognitive and motoric subsystems. Assuming that mechatronic systems are the formation of function tree, each function is fulfilled by the typical sensory, motoric, cognitive subsystems. Those sub systems compromises a mechatronic module. If we consider the dish machine example; transitions represent subfunctions or activities of a mechatronic system, places represent the sensory information coming from environment and decision functions represent processor which possesses the sensory information coming from the environment.

CHAPTER 4

THE DESIGN NETWORK SIMULATOR FOR THE IMPROVED N-PNDN THEORY

The Design Network Simulator (Güroğlu, 1999) is further developed for the N-PNDN theory which enables us to decompose and represent a mechatronic product in multi level form. As mentioned before each level is based on PNDN theory and independent of each other.

In this chapter an overview of the software package will be given. Furthermore the algorithm developed for the construction of N-PNDN and the algorithm for the token flow of N-PNDN will be presented.

4.1 AN OVERVIEW OF FURTHER DEVELOPED DNS

4.1.1 Programming Environment

The visual programming environment of Design Network Simulator makes it possible to decompose the mechatronic systems into its functional states. The integrated development environment (IDE) of Borland C++ Builder is used during the development of the program. The fundamental elements of N-PNDN (transitions, place sets, I/O Mapping and switches) are designed as separate elements, and creating new layer for each transition is enabled. For convenience, the above mentioned features are combined and the user does not need any additional programming.

4.1.2 Structure of the Software Package

DNS is composed of two parts.

- Graphical User Interface
- Software Modules

Graphical User Interface

Graphical User Interface consists of the design window, the tool bar and the control panel.

Software Modules

Software modules are subdivided into three parts which are shown in the Figure 4.1. The creation and simulation of N-PNDN is also involved in modules and will be investigated in the next section of this chapter.

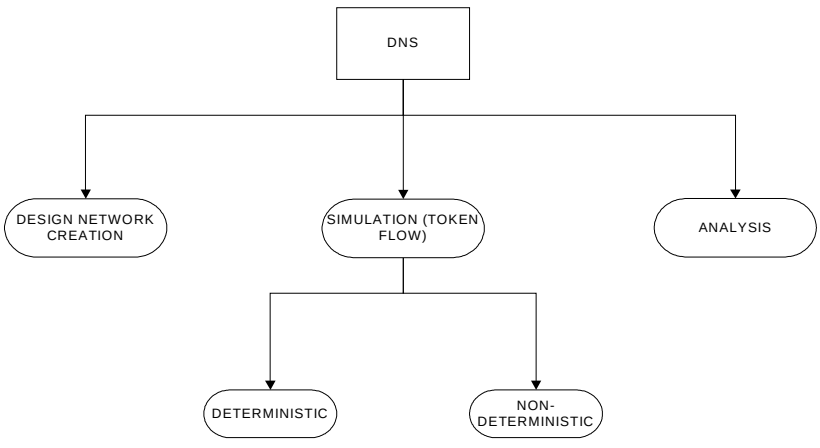


Figure 4.1 Architecture of Design Network Simulator.

4.2 GRAPHICAL USER INTERFACE

The Graphical User Interface should support both for single and multi level network creation. In addition to this, it should also support storing, retrieving and printing designs. GUI includes design window, object inspector, tool bar, control panel, help part.

Design Window

Design window is a basic form which is provided by Borland C++ Builder IDE (Integrated Development Environment). In this form the user can place the transitions, place sets, switches and create links between the net elements in order to create the design network.

When it is necessary the user can also change the locations of the elements of net by dragging. For the further applications, the created design network can be saved as text file in a directory and can be regenerated by retrieving it.

Furthermore DNS allows to print the design network and those basic features are all combined in Design Window. It is also possible to divide the design window into grids and therefore the user can align the elements of the design network. This provides a more tidy structure. When the user clicks an element to put on the design window, the upper left corner of the element is attached on the selected grid.

When the program first executes, the base form appears on the screen with control panel and tool bar. The user should open a new design window or retrieve a design which is saved before. At the same time object inspector window also appears near to the design window in order to inform the user about the elements.

In Figure 4.2 the screen shot of the main screen of DNS is given.

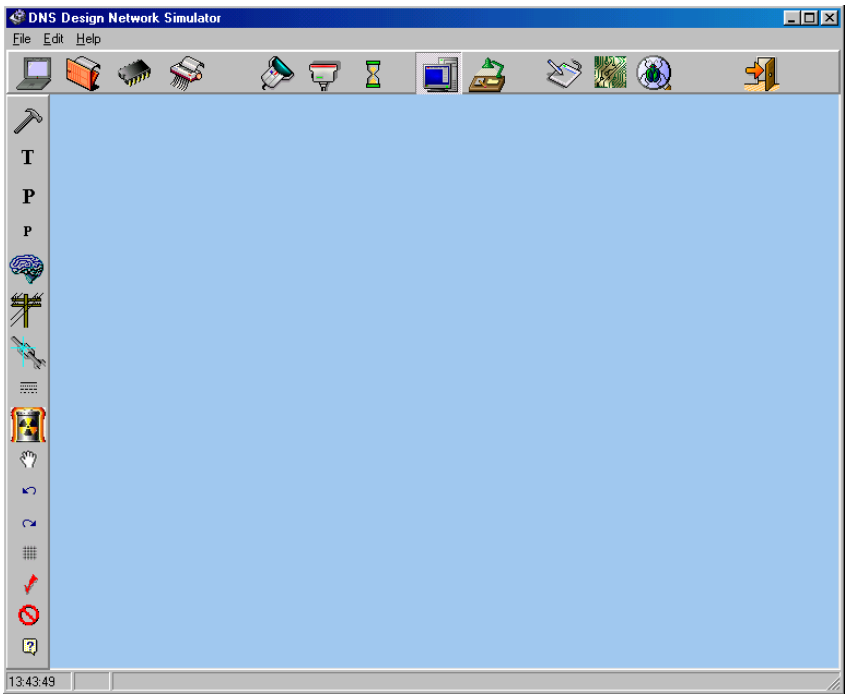


Figure 4.2 Graphical User Interface

Object Inspector

Object Inspector window is used for data access for the network elements. The user can easily edit or enter the information for the related network element. Figure 4.3 shows the screen shot of the object inspector window for the on off button component of dish machine.

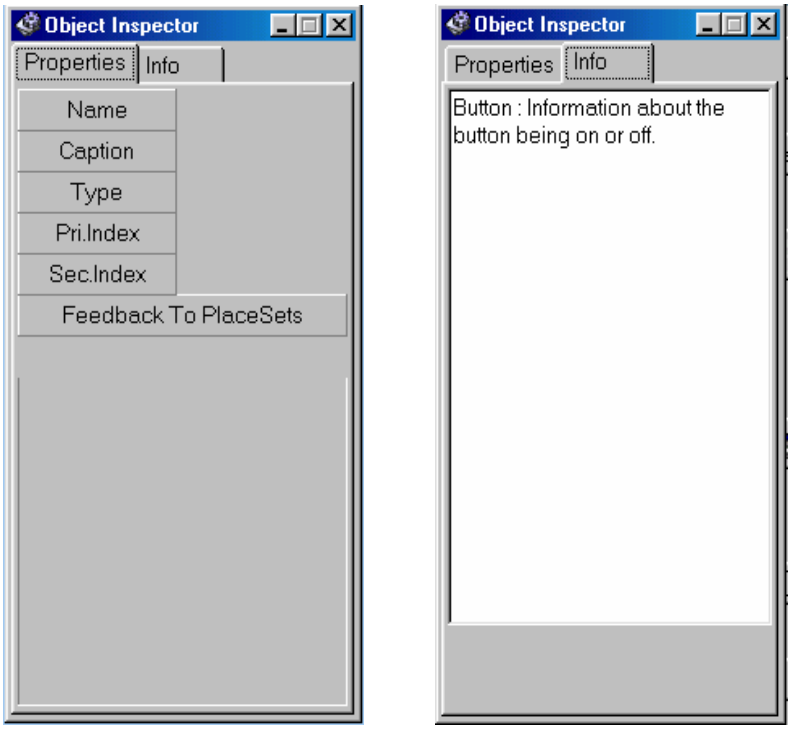


Figure 4.3 Object Inspector of DNS

The first row of the object inspector window indicates the name of the component clicked on. In the second row, for convenience, the caption can be replaced by an abbreviation which makes easy to inform the user. The third row determines the type of the component such as Transition, Place Set and Switch and is defined automatically. Any attempt to change the name, number and sub number property of the network components are not allowed. The primary index which has been given in the fourth row is only defined for the transitions (functional states), place sets (variables and instantiations), switches (decision functions). On the other hand the secondary index is only defined for the place sets of the design network and used to distinguish the variables and the instantiations. Info tab of the component is used for the information access and that access is done by text box.

Tool Bar

The tool bar is the part on which the buttons are offered for the network construction operations and located on the left hand side of the design window. Figure 4.4 shows the tool bar and its buttons.

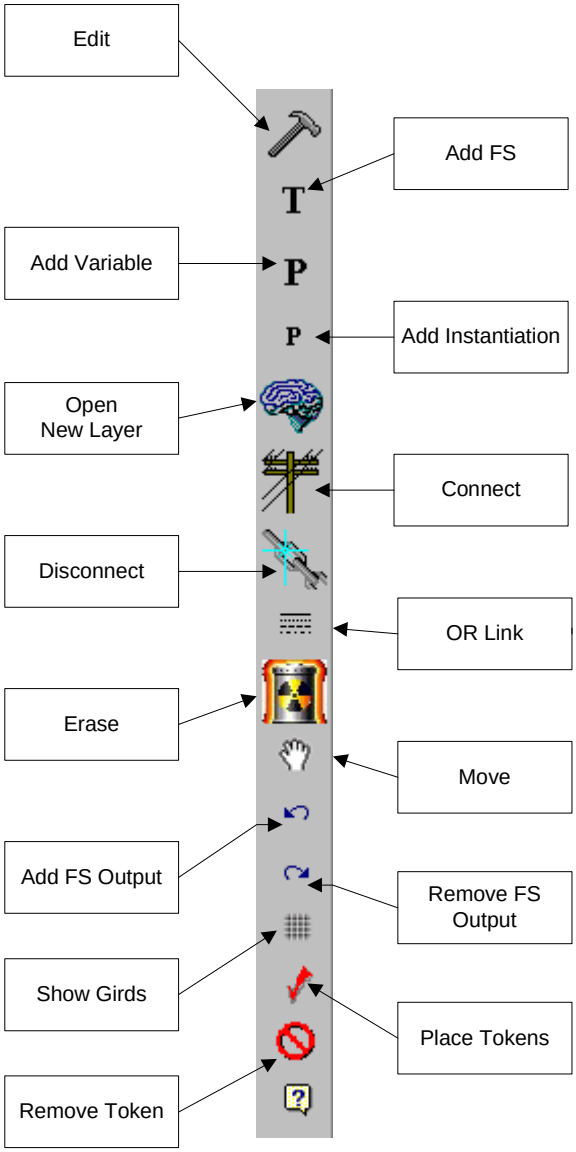


Figure 4.4 . The tool bar of the DNS

The edit button calls for the object inspector window and enables the user to do the events that are explained in the previous part. The basic components of design network are created by clicking once on functional state, add variable, add instantiation button. In order to create a new layer to the selected transition the user should click on the open new layer button. Connect, Or link and Disconnect buttons are used for creating and removing input and output mappings between the components of the design network. Erase button is used in order to delete a component. Once a component is deleted, it is impossible to recall the deleted item. Move button is for dragging the selected items to the desired location of the design window. If we want to establish or remove a functional output then one should click on the Add FS output or Remove FS output buttons respectively. As mentioned before if the user wants to divide the design window into grids, one should click on the grid button. During the simulation the token insertion is done by clicking on the place token button and vice versa is done by clicking on the remove token button.

Control Panel

Control panel presents the basic facilities to the user like new, open save, print commands. Other than these commands are explained below. The Figure 4.5 shows the control panel of DNS.

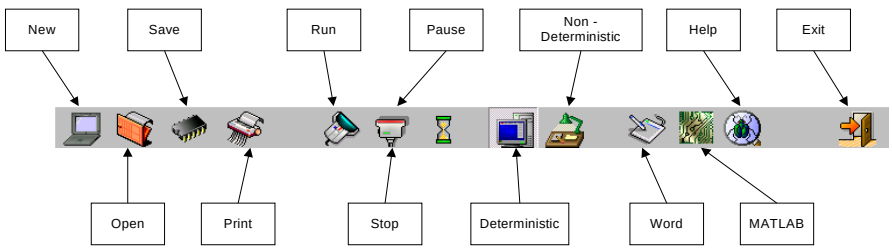


Figure 4.5 – The control panel of DNS

As shown above, the first group indicates the “new”, “open”, “save”, “print” buttons. Later on “run”, “stop” and “pause” buttons come for which, one uses during the simulation and therefore token flow.

Deterministic and non-deterministic buttons determine the simulation type and also indicate which type of simulation is valid through the run time of program. The MsWord and MatLab icons open those program interfaces. Exit button terminates the program execution.

4.3 SOFTWARE MODULES OF DNS

4.3.1 Creation of Design Network

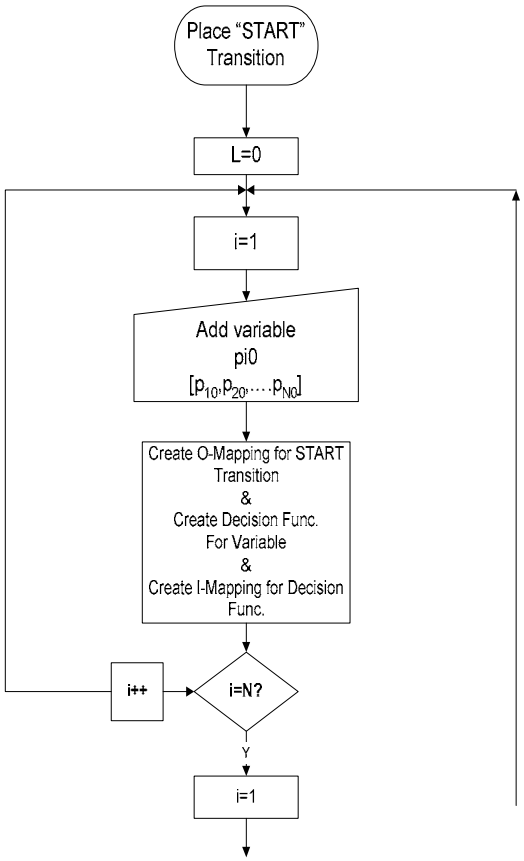
Creation of design network starts with placement of the start transition on the design window. The caption of the first transition is “START”. The second step is place the variables. When the user clicks on the variable button and then clicks on the design window a variable is automatically created. At the same time the O-Mapping of the START transition is also established by DNS.

In addition to that, having placed the variables of the design network, decision functions of the related variables are also placed with their I-Mappings automatically.

In the following step one should define the instantiation of the variables and O-Mapping for decision functions. When this is done, the O-Mappings for decision functions are also created automatically.

As last step, by placing the other functional states which are represented by transitions and after creating the following I/O Mappings, the first layer PNDN network is completed.

Now if one wants to expand and decompose a functional state, “Open NewLayer” button should be clicked. When this is done, the valid design window is saved as text file and then removed from stack in order to avoid stack overflow. Later on DNS opens a new design window and makes it possible to create a new PNDN network for the relevant functional state. In this case a new button with a caption “BACK” appears on the left bottom corner of the new created design window. The user can return to the one upper level by clicking this button when he desires. The point on which the user should give his attention that; a “STOP” transition in each sub level should be created since the aim is to return the token to the upper level during simulation. The algorithm for the creation of N-PNDN is given Figure 4.6:



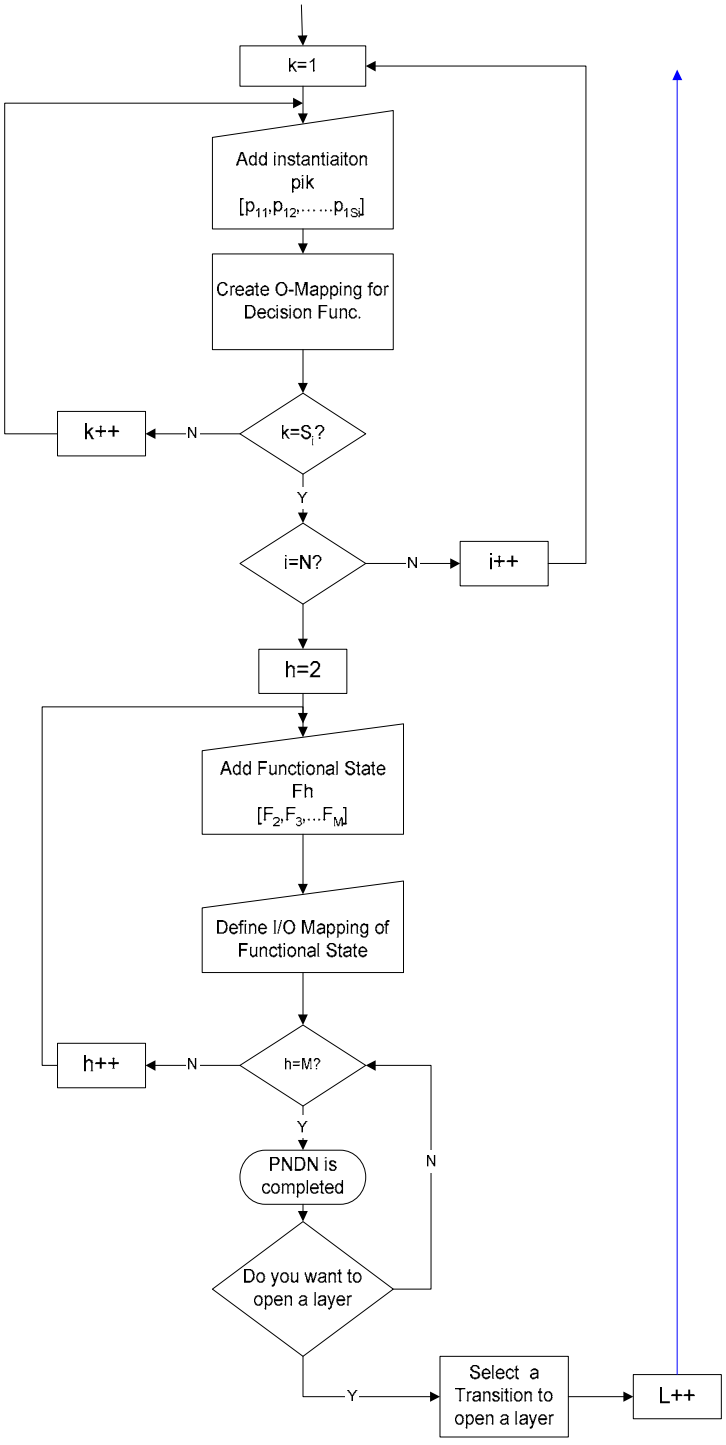


Figure 4.6 Algorithm for the Creation for N-PNDN

In this algorithm;

i is a counter for Variables

N is the number of Variables

K is a counter of Instantiations

S_i is the number of Instantiations for the i^{th} Variable

H is a counter for Functional States,

M is the number of Functional States.

L is the counter for level

The algorithm is applied to the dish machine example. As mentioned before the first step is to place the "START" transition on the design window. The O-Mapping of START transition; Decision Functions and I-Mappings for decision functions are created automatically.

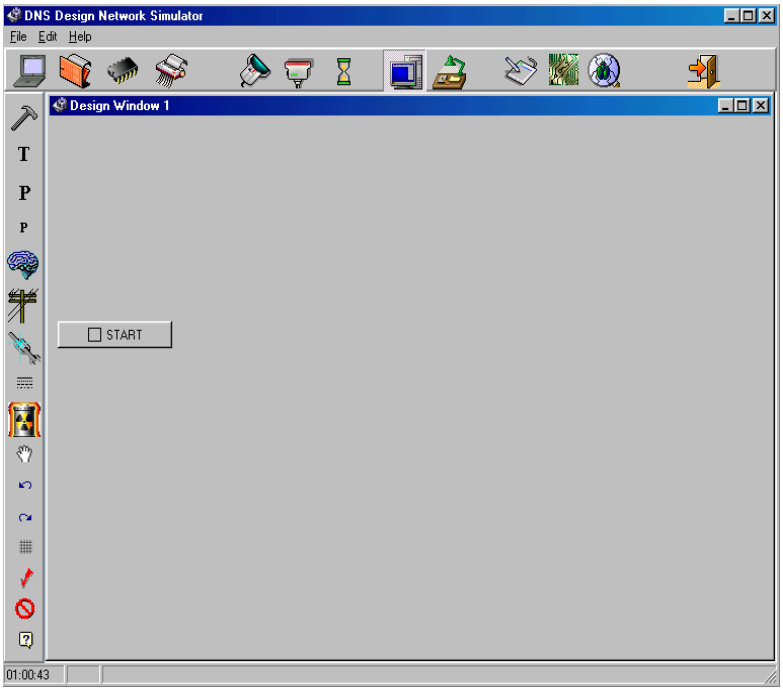


Figure 4.7 Creation of N-PNDN for the dish machine

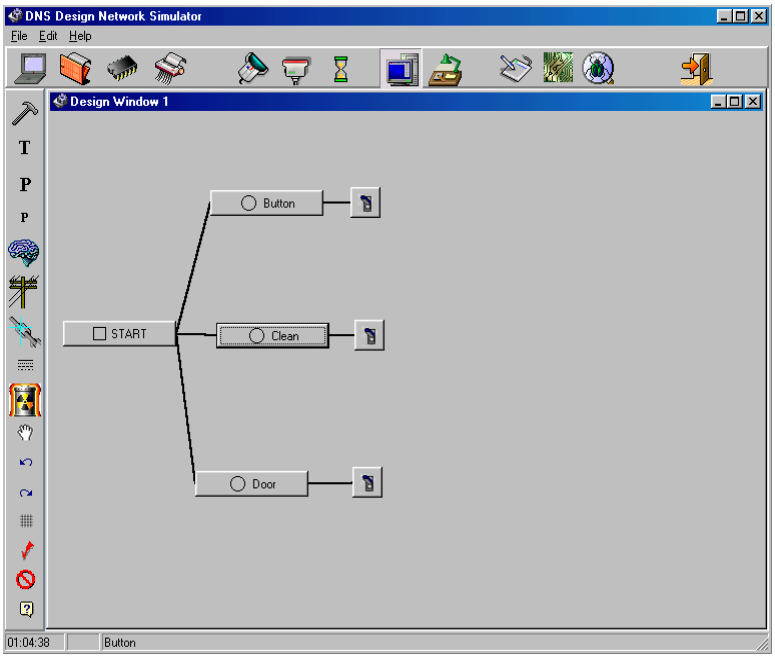


Figure 4.8 Creation of N-PNDN for the dish machine (continued)

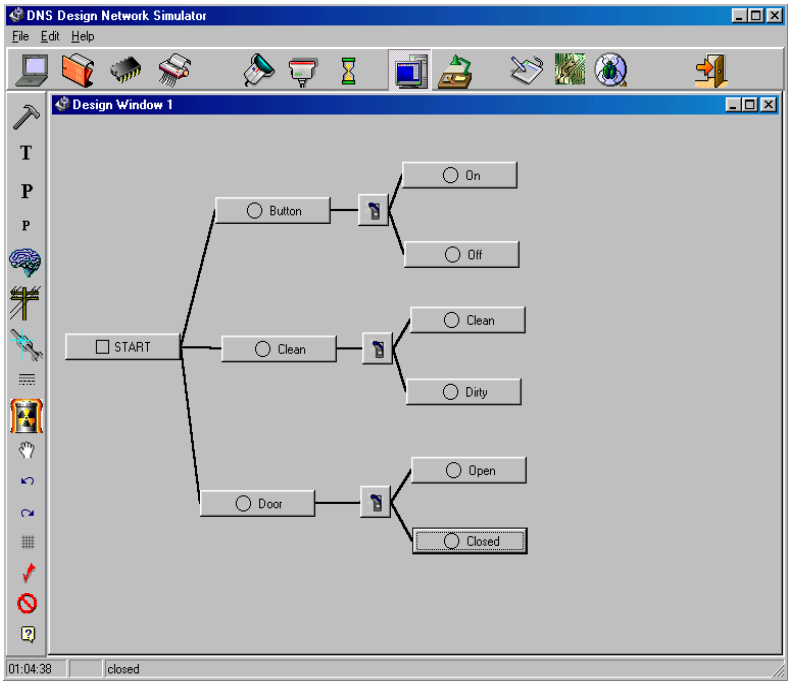


Figure 4.9 Creation of N-PNDN for the dish machine (continued)

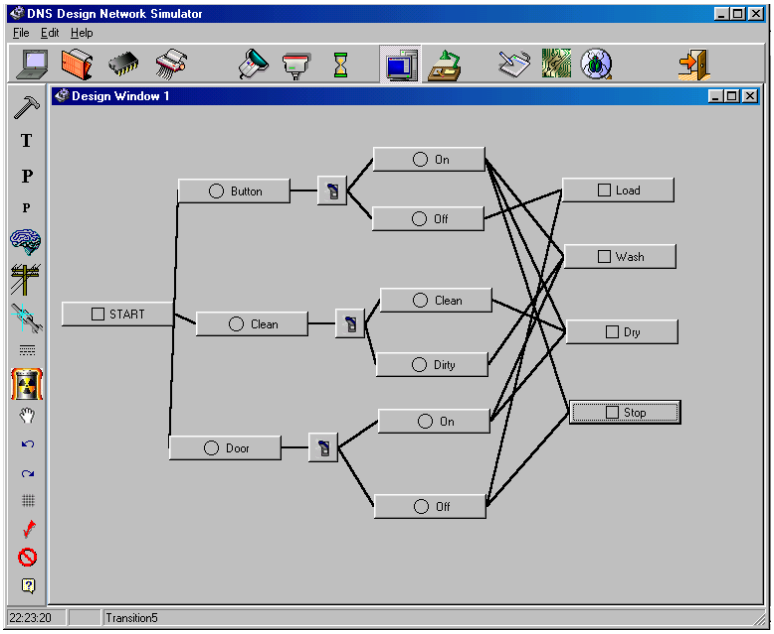


Figure 4.10 Creation of N-PNDN for the dish machine (continued)

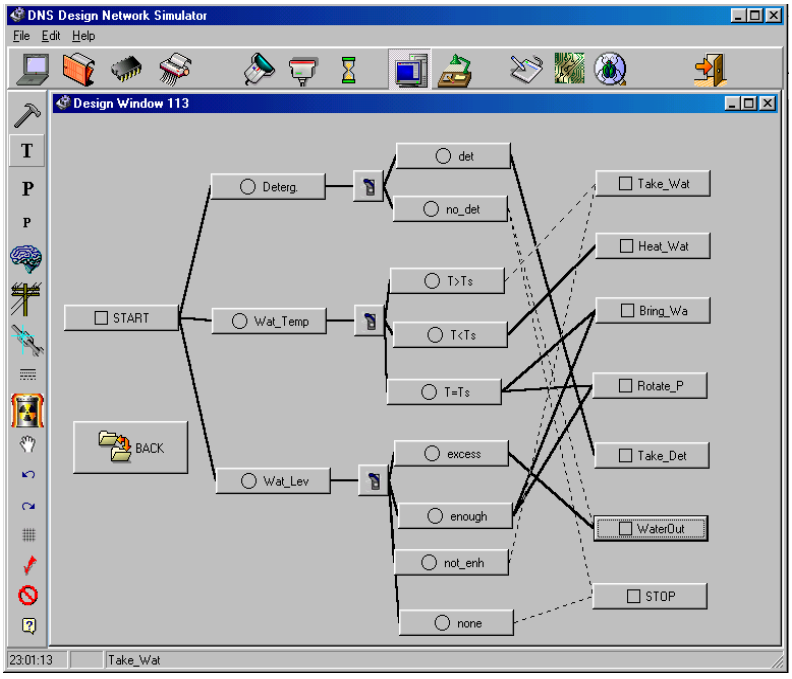


Figure 4.11 Creation of N-PNDN for the dish machine (continued)

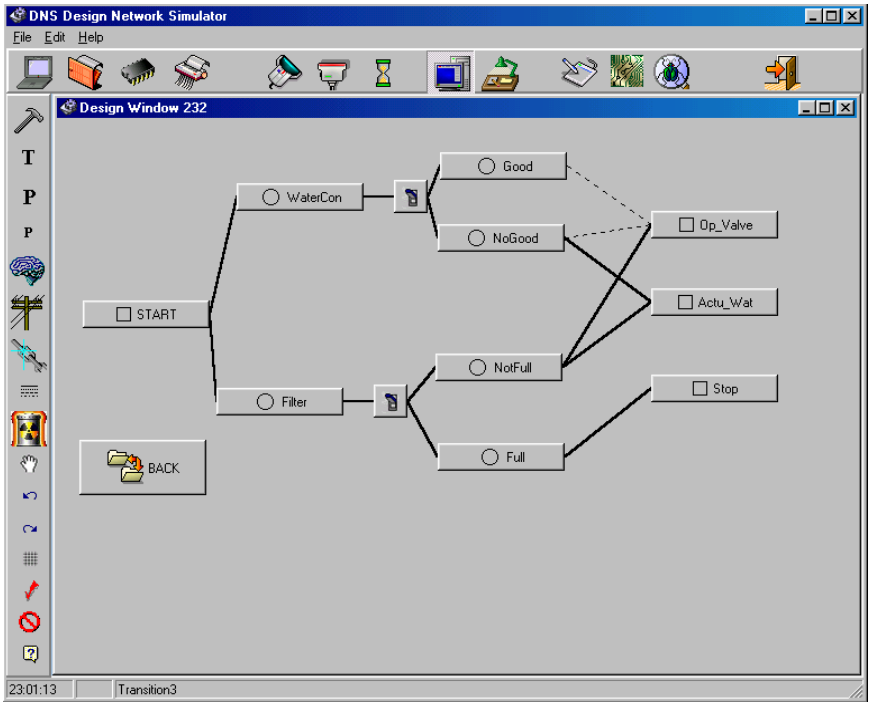


Figure 4.12 Creation of N-PNDN for the dish machine (continued)

Figure 4.7–4.12 illustrates the creation of N-PNDN modeling of dish machine example. As it can be seen easily each time when a new level is created a button with a caption “BACK” is automatically created in order to return to the upper level during construction.

4.3.2 Token Flow in N-PNDN (Simulation)

Having modeled the product in N-PNDN, it is easy to simulate and model the dynamic behavior of the design. The main purpose in modeling is the information flow between the network elements. The DNS presents two types of token flow. Those are deterministic and non-deterministic token flows. DNS enables both type of token flow for the single level decomposition. However the developed algorithm supports only deterministic token flow for the multi-level decomposition of mechatronic products.

Deterministic Token Flow

The new developed algorithm for the deterministic token flow is given below in Figure 4.10. The simulation begins with placing the tokens on the place sets. This variable marking, M^V, enables the decision functions. When the token flow starts, enabled decision functions fire and tokens are re-placed in one of the instantiations of their variables.

After the instantiation marking, the functional states which have tokens in their instantiations are enabled. This means that the enabled transitions can fire during the token flow.

When one or more of the transitions have sub levels, the design window that carries the PNDN structure of that transition opens and the present design window is saved and deleted from stack. As the new design window opens, the token passes to the start transition of the sub level and the token flow is restarted by the user. At this stage, the back button doesn't appear since the token flow is active and DNS does not allow the user to return to the upper level.

In the sub level of DNS, the token flow goes on till the stop transition fires and the PNDN rules prevail also in this level. The firing of "STOP" transition means that the function of the decomposed transition is finished and the token can pass to the upper level. If the existing transitions have also sub levels in the active window, the process goes on till the token visits all the stop transitions of the relevant level. The N-PNDN structure is time independent. Therefore the token, flows only through the active design window.

When the stop transition fires, the active design window is saved as text file by DNS. The token passes to the upper level and places again in the

transition where token flow is lastly interrupted because of the sub level decomposition of that transition.

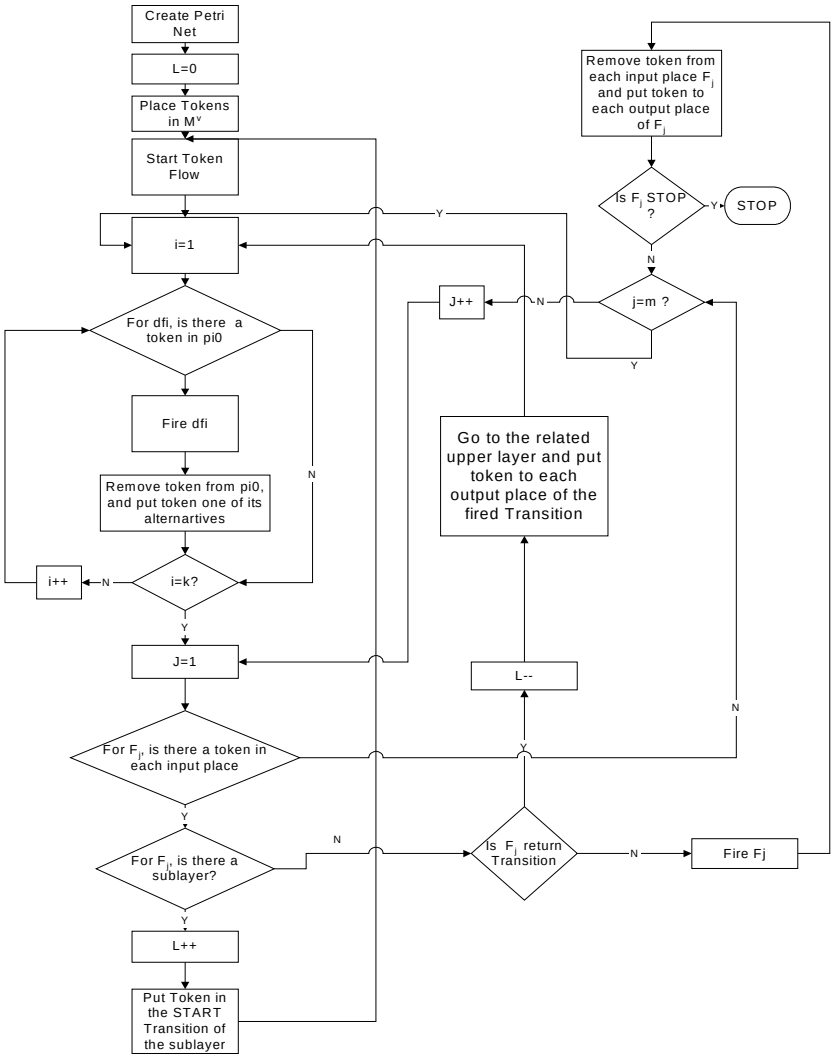


Figure 4.13 The algorithm for Deterministic Token Flow in N-PNDN

Figure 4.13 shows the deterministic Token Flow on N-PNDN of dish Machine and the firing of “START” transition. The user should place the token in the variables which is called variable marking.

The user places the tokens on the selected variables. Figure 4.14 shows that the variables “BUTTON”, “CLEAN” and “DOOR” have tokens and the DNS gives the message of “Please place the tokens on instantiations”.

After this stage as it is shown in Figure 4.14, DNS is waiting the user to place the tokens on one of its instantiations. The instantiations of “ON”, “DIRTY” and “CLOSED” are selected by placing tokens on them in order to fire the “WASH DISHES” functional state. As it is mentioned in previous chapter the “WASH DISHES” functional state has multi level decomposition.

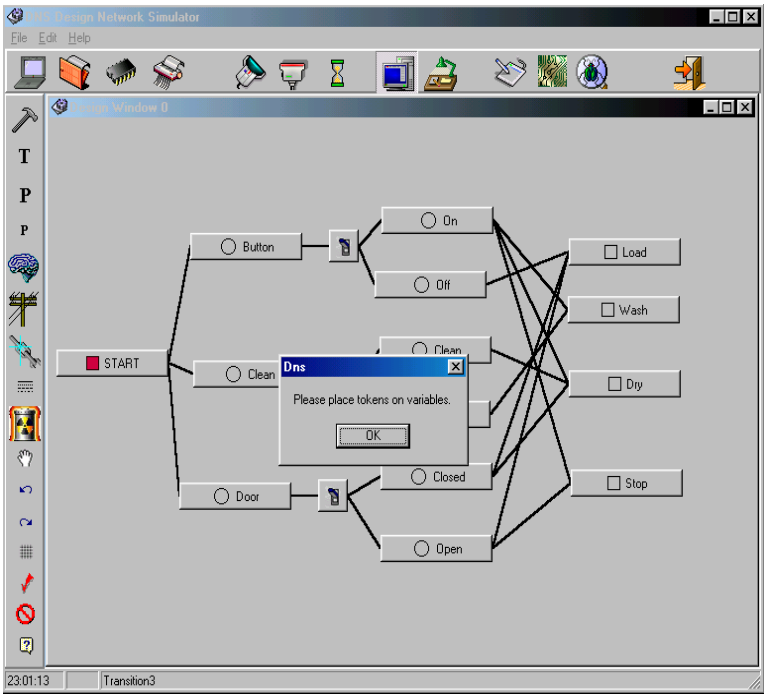


Figure 4.14 Token Flow in N-PNDN

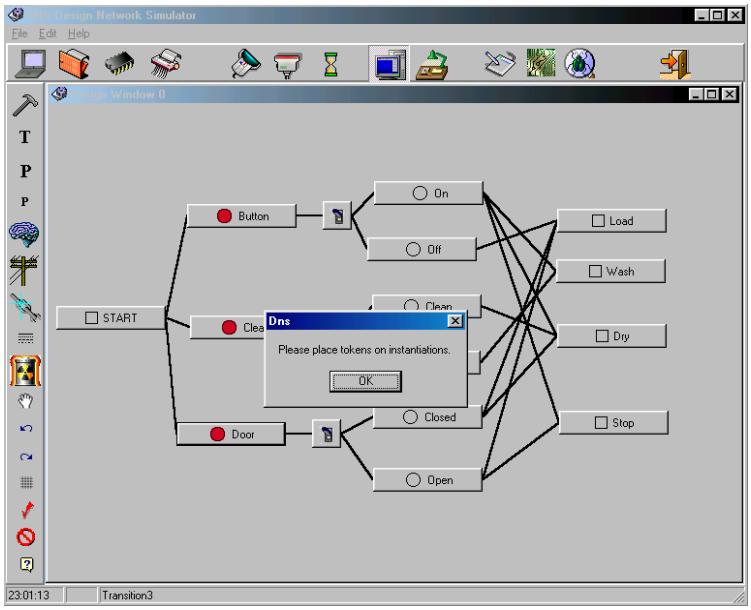


Figure 4.15 Token Flow in N-PNDN (continued)

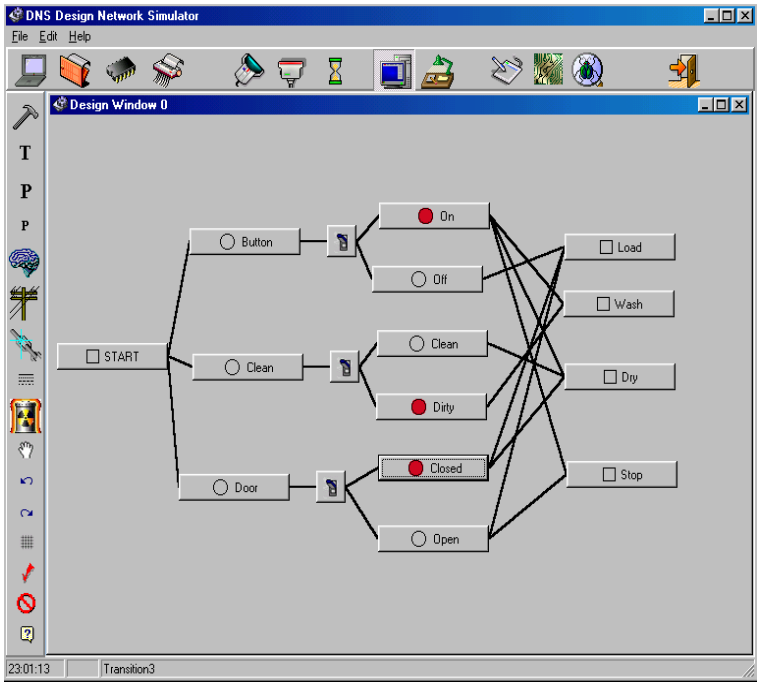


Figure 4.16 Token Flow in N-PNDN (continued)

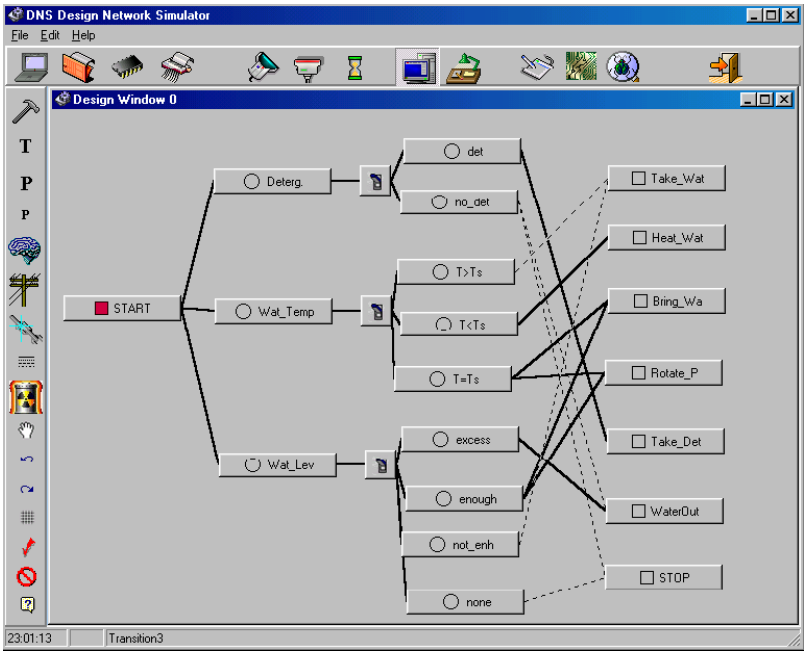


Figure 4.17 Token Flow (continued)

When the functional state of "WASH DISHES" is fired, the new design window and the sub level of the functional state are automatically retrieved as seen in Figure 4.17. Now the user should start the token flow again. This is done by clicking the run button and a token is placed on the "START" transition. Therefore the token flow starts for the "WASH DISHES" functional state as shown in Figure 4.17

Then DNS waits the user to place the tokens on the variables and gives the related message. After placing the tokens on variables the next step is again to place the tokens on the desired instantiations as shown in Figure 4.18.

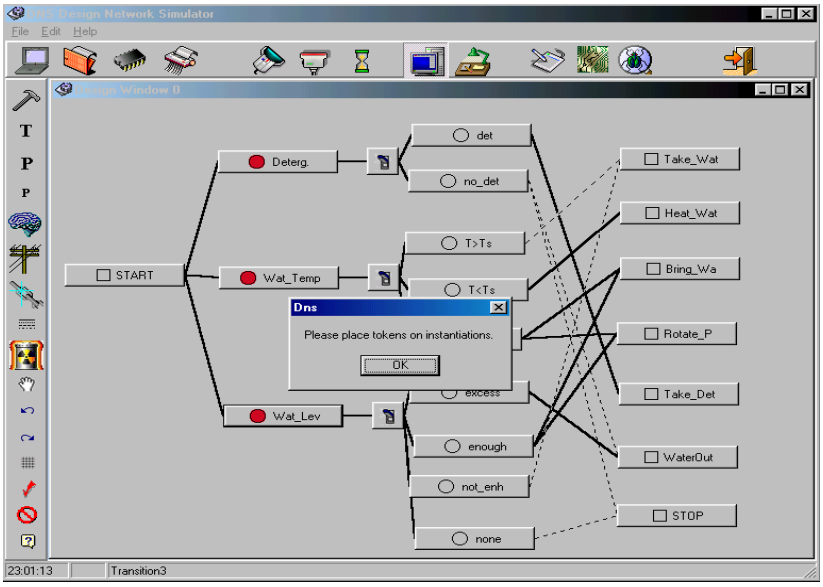


Figure 4.18 Token Flow (continued)

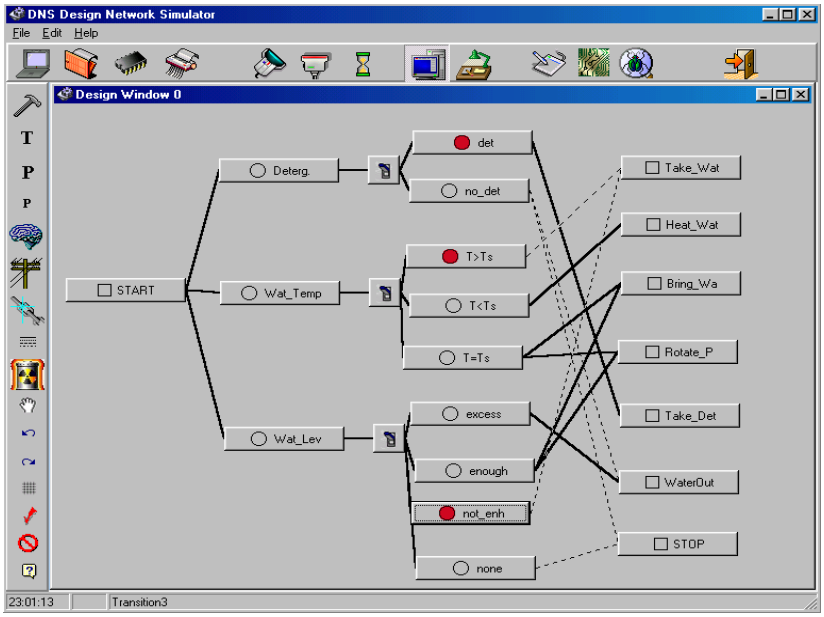


Figure 4.19 Token Flow (continued)

The instantiations of “DET”, “T>Ts” and “NOT ENOUGH” are selected in order to fire the “TAKE WATER” functional state as shown in Figure 4.19. The “TAKE WATER” functional state has sub level and the information

flow will go on till the token reaches to the final resolution level of the related functional state.

When the “TAKE WATER” functional state is fired the active design window is closed and the sub level of “TAKE WATER” functional state is opened. The PNDN structure of that functional state is given in Figure 4.20 and the user should start the token flow and therefore the information flow by placing the token in “START” transition.

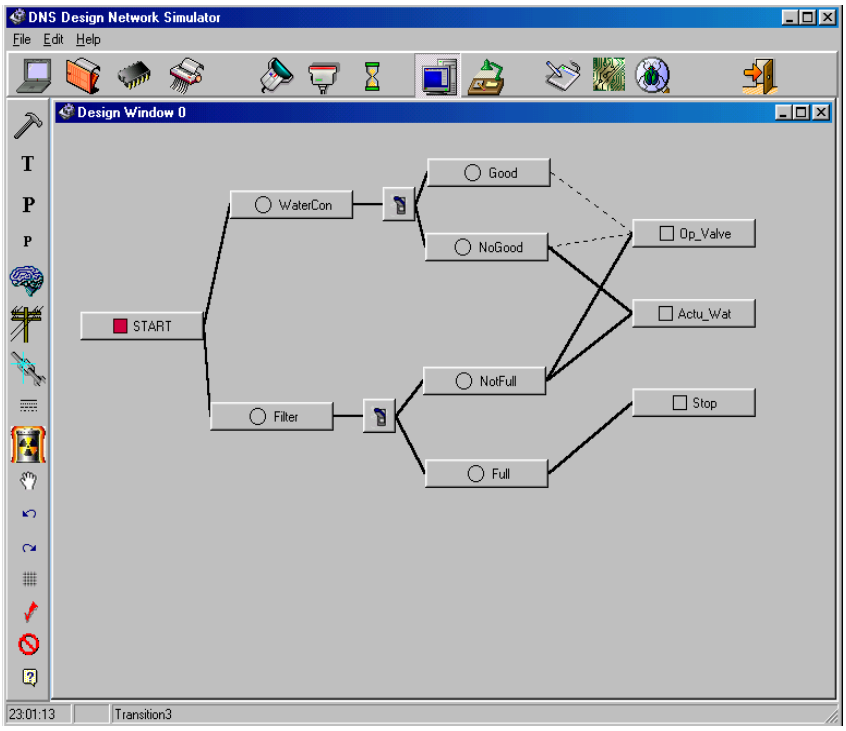


Figure 4.20 Token Flow (continued)

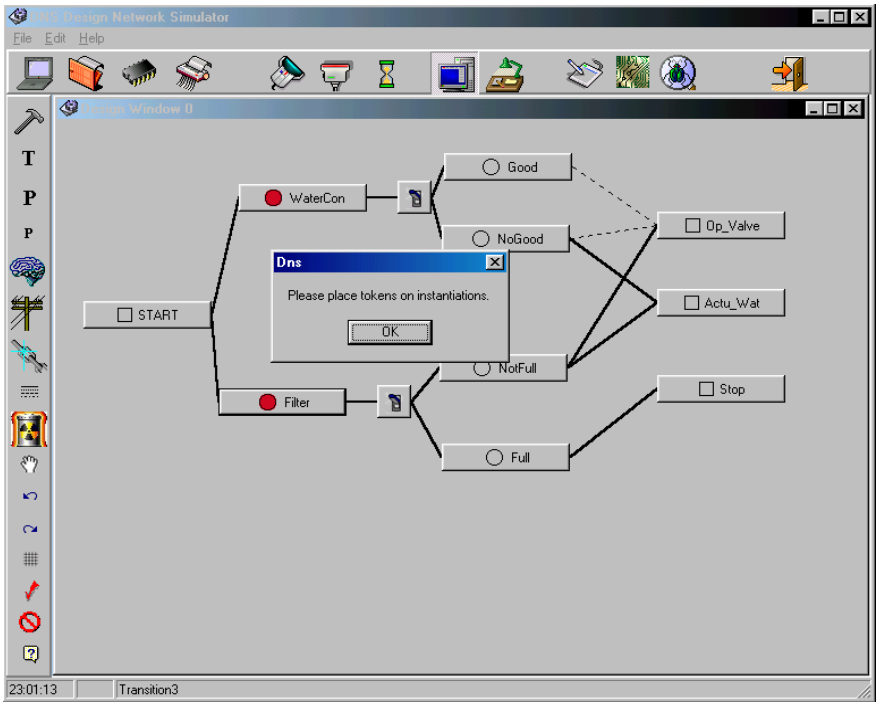


Figure 4.21 Token Flow (continued)

After placing the tokens on the "WATER CONDITION" and "FILTER" place sets, the DNS waits the designer to place the tokens on one of the instantiations of the relevant place sets as shown in the Figure 4.21.

In Figure 4.22, having placed the tokens on the instantiations, the "STOP" transition is fired in order to finish the token flow and therefore the information flow for the "TAKE WATER IN" functional state.

When the STOP transition is fired the active design window is closed which indicates the end of token flow for the active design window.

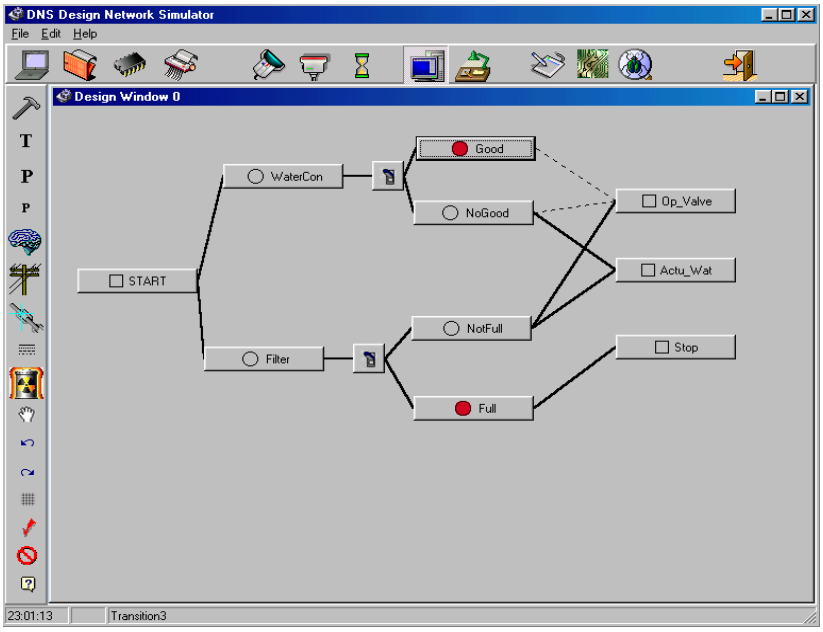


Figure 4.22 Token Flow (continued)

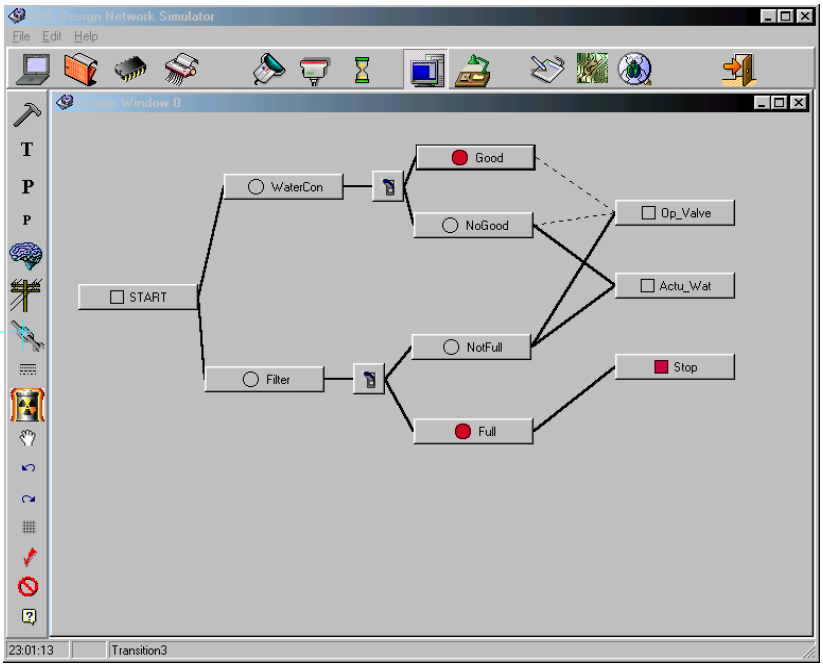


Figure 4.23 Token Flow Figure (continued)

In Figure 4.24 the upper level and therefore the PNDN structure of “WASH DISHES” is opened.

Now DNS is waiting for the user again to place the tokens in one the instantiation of the place sets and so the token flow restarts.

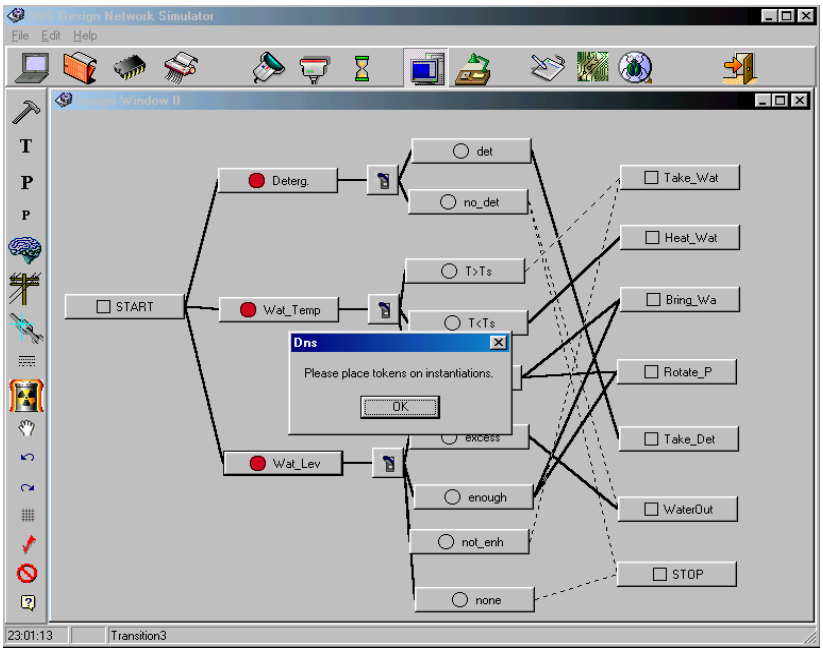


Figure 4.24 Token Flow

CHAPTER 5

CASE STUDIES

In this chapter 4 case studies have been given and note that FDT is dependent on the designer. This means that different designers can form different functional models which yields different physical systems based on decisions made by different designers.

5.1 N-PNDN Model of Mouse

The N-PNDN model of mouse has “Change Position of Cursor in 2D”, “Detect and measure changes in X & Y”, “Send Data”, “Apply Control Gain”, “Stop” transitions in the first level of decomposition.

The functional state of “Detect and measure changes in X & Y”, is decomposed as “Generate light pulses”, “Convert into X and Y movement” and “Stop” transition in the second level of decomposition. “Control Gain” functional state is decomposed as “D/A Conversion” and “Apply Gain” and “Stop” transition. Finally the functional state “Send Data” is decomposed as “A/D Conversion”, “Generate IR Signal”, “Receive IR signal” and “Stop” transition. The functional design tree of mouse is given in Figure 5.1 and the following computer implementation of mouse is given in Figure 5.2 and Figure 5.3.

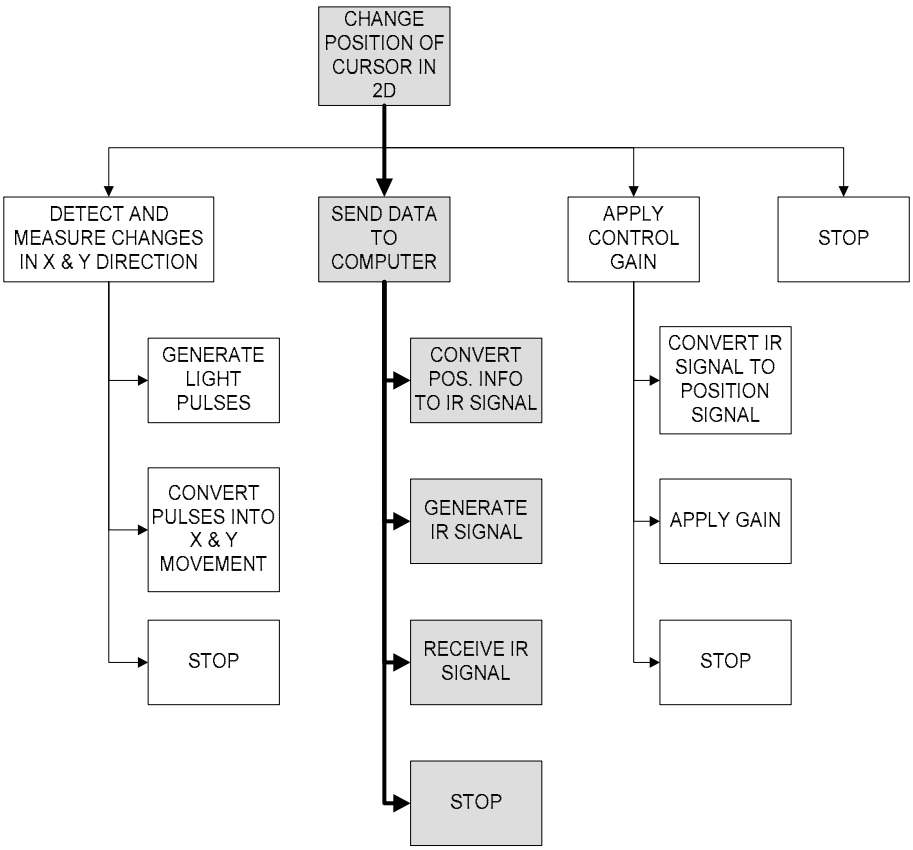


Figure 5.1 Functional Design Tree of Mouse

Definition of Functional States

FIRST LEVEL:

F (S) = { F_1 , F_2 , F_3 , F_4, F_5 }

F1 = Change Position of Cursor in 2D,

F2 = Detect and measure changes in X & Y,

F3 = Send Data (distance / direction / speed) to Computer

F4 = Apply Control Gain

F5 = Stop

SECOND LEVEL

Functional decomposition of the functional state “F2-Detect and measure changes in X & Y”

F (S) = {F_1, F_2, F_3, F_4}
F1= Generate Light Pulses
F2 = Convert Pulses into X and Y movement
F3 = Stop

The decomposition of the functional state “F3-Send data distance/direction/speed to Computer” as follows:

F (S) = {F_1 , F_2 , F_3 , F_4}
F1= Analog/Digital Conversion (Convert Pos. Info to IR Signal)
F2 = Generate IR signal
F3 = Receive IR signal
F4 = Stop

The decomposition of the functional state “F4-Apply Control Gain” as follows:

F (S) = { F_1 , F_2 , F_3)
F1= Digital/Analog Conversion (Convert IR Signal to Position Signal)
F2 = Apply Gain
F3 = Stop

Definition of Variables

FIRST LEVEL

p_{10} = movement (information about existence of movement)
p_{20} = RS data (information about information in serial port)

SECOND LEVEL

The variables for the sub-level of the functional state “Detect & Measure”:

p_{10} = motion movement (information about existence of motion)
p_{20} = pulse check

The variables for the sub-level of the functional state “Send Data”:

p_{10} = Measurement Info
p_{20} = pulse check (the information about the existence of the pulse)

The variables for the sub-level of the functional state “Control Gain”:

p_{10} = signal (is there any signal or not)
p_{20} = Gain (the amount of gain to be applied)

Instantiations of Variables

FIRST LEVEL

p_{11} = mov (there is movement
p_{12} = nomov (there is no movement)
p_{21} = data (there is information in serial port)
p_{22} = nodata (there is no information in serial port)

SECOND LEVEL

The instantiation variables for the sub-level of the functional state “Detect & Measure”:

p_{11} = motion (motion is detected)
p_{12} = no motion (no motion is detected)
p_{21} = Exist (pulse exist)
p_{22} = No Exist (no pulse exist)

The instantiation variables for the sub-level of the functional state “Send Data”:

p_{11} = info (there is info)
p_{12} = no-info (there is no info)
p_{21} = signal (there is signal)
p_{22} = no-signal (there is no signal)

The instantiation variables for the sub-level of the functional state “Control Gain”:

p_{11} = signal exist(there is signal)
p_{12} = no-signal (there is no signal)
p_{21} = gain 1 (apply gain G1)
p_{22} = gain 2 (apply gain G2)
p_{23}= gain 3 (apply gain G3)

Decision Functions

FIRST LEVEL

$DF=\{df_1 , df_2 \}$

df_1= decision function for the movement variable
df_2= decision function for the serial port

SECOND LEVEL

The decision functions for the sub level of the function state "Detect & Measure" follows:

$DF=\{df_1, df_2\}$

df_1= decision function for the motion variable
df_2= decision function for the pulse check

The decision functions for the sub level of the function state "Send Data" follows:

$DF=\{df_1, df_2\}$

df_1= decision function for the measurement info variable
df_2= decision function for the signal variable

The decision functions for the sub level of the function state “Control Gain” follows:

DF={df_1, df_2}

df_1= decision function for the measurement info variable
df_2= decision function for the signal variable

DEFINITION OF I – MAPPINGS

FIRST LEVEL

I (movement, df_1) = 1 I(data, df_2) =1
I (mov, F_1) = 1 I (nomov, F_4) = 1
I (data, F_3) = 1 I (nodata, F_4) = 1 I (mov, F_2) = 1

SECOND LEVEL

The I-Mappings in the sub level of the functional state of “Detect & Measure”:

I (motion, df_1) = 1 I (pulse check, df_2) =1
I (motion, F_1) =1 I (notexist, F_1) =1
I (exist, F_2) =1 I (no motion, F_3) =1

The I-Mappings in the sub level of the functional state of “Send Data”:

I (measure info, df_1) = 1 I(signal, df_2) =1
I (info, F_1) = 1 I (no signal, F_2) = 1
I (info, F_2) = 1 I (signal, F_3) = 1

I (no info, F_4) = 1

The I-Mappings in the sub level of the functional state of "Control Gain":

I (signal, df_1) = 1 I (gain, df_2) =1
I (signal exist, F_1) = 1 I (signal exist, F_2) = 1
I (gain 1, F_2) = 1 I (gain 2, F_2) = 1
I (gain 3, F_2) = 1 I (no signal, F_3) = 1

DEFINITION OF O – MAPPINGS

FIRST LEVEL

O (df_1, mov) = 1 O(df_2, data) =1 O (F_3, movement) = 1
O (df_1, nomov) = 1 O(df_2, nodata) = 1 O(F_3, RSdata) = 1
O(F_1, movement) = 1 O(F_2, movement) = 1 O(F_4, movement) = 1
O(F_1, RSdata) = 1 O(F_2, RSdata) = 1 O(F_4, RSdata) = 1

SECOND LEVEL

The O-Mappings of sub level of the functional state of "Detect & Measure":

O (df_1, motion) = 1 O(df_1, no motion)=1 O (df_2, Exist) = 1
O(df_2, no exist)=1 O(F_1,Motion) = 1 O(F_2,Pulse Check) = 1
O(F_1, Pulse Check) = 1 O(F_2, Motion) = 1

The O-Mappings of sub level of the functional state of "Send Data":

O (df_1, info) = 1 O(df_1, no info)=1 O (df_2, Signal) = 1
O (df_2, no signal) = 1 O(F_1,Measure Info)=1 O(F_2,Measure Info)=1

$O(F_3,\text{Measure Info})=1$ $O(F_1,\text{Signal})=1$ $O(F_2,\text{Signal})=1$
$O(F_3,\text{Signal})=1$

The O-Mappings of sub level of the functional state of “Control Gain”:

$O(df_1,\text{signal}) = 1$ $O(df_1,\text{no signal}) = 1$
$O(df_2,\text{gain 1}) = 1$ $O(df_2,\text{gain 2}) = 1$ $O(df_2,\text{gain 3}) = 1$
$O(F_1,\text{A signal}) = 1$ $O(F_2,\text{Signal}) = 1$
$O(F_1,\text{gain }) = 1$ $O(F_2,\text{gain}) = 1$

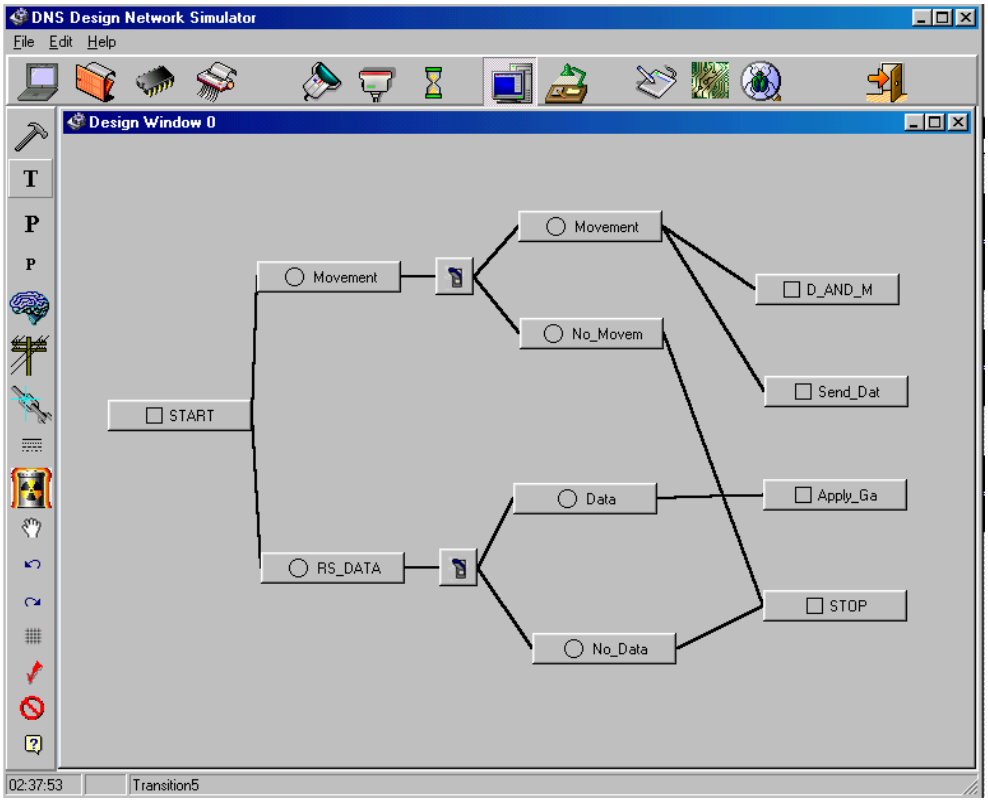


Figure 5.2 PNDN of Mouse at the first level decomposition

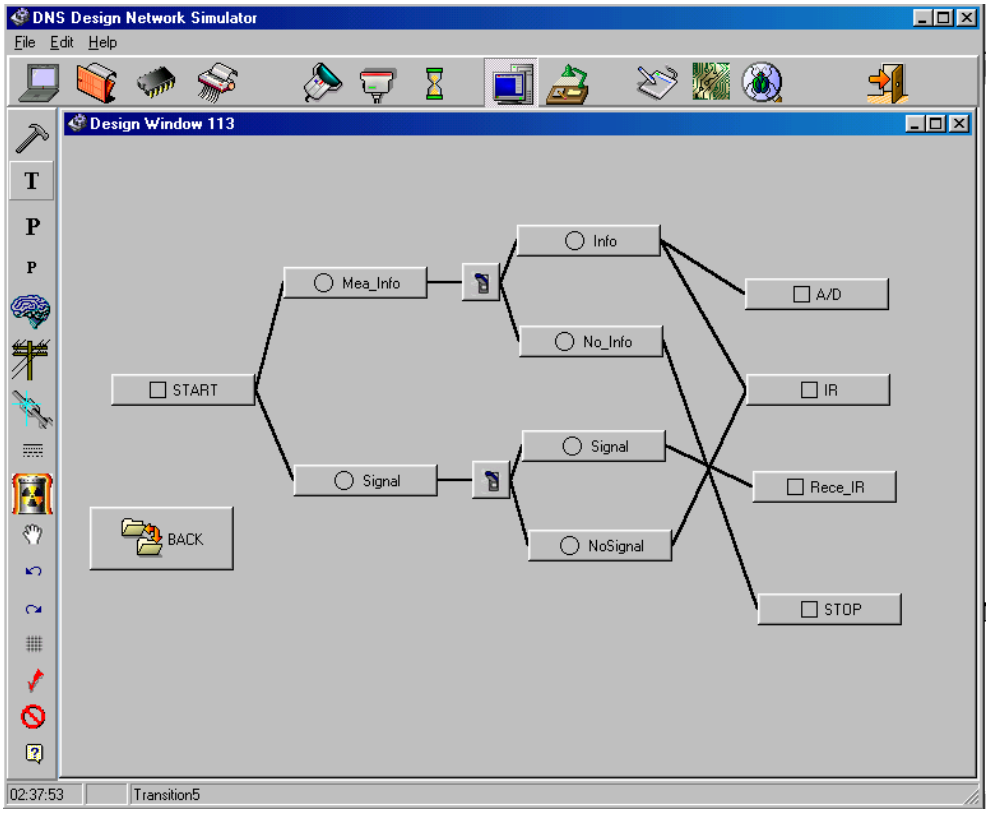


Figure 5.3 PNDN model of Mouse for the second level

5.2 N-PNDN Model of CD player

In the N-PNDN model of CD player “Start”, “Import CD”, “Rotate CD”, “Read Data on CD”, “STOP/ Export CD” come in the first decomposition level. The functional state “Import CD” is decomposed to “Import Electricity”, “Actuate Electricity” and “Convert Electricity” and “Stop” functional states in the second level. In addition to that, the functional state in the first level “Rotate CD” is decomposed into “Import Electricity”, “Convert Electrical energy into Mechanical Energy” and “Stop” functional states. The functional state in the first level “Read Data on CD” is decomposed into “Import Electricity”, “Convert Electrical Energy into Optic Energy”, “Send out Optic Energy”, “Collect the Reflected Energy”, “Analog to Digital Conversion” and “Stop” functional states in the second

level. Lastly the functional state “Export CD” in the first level is decomposed into “Import Electricity”, “Actuate Electricity”, “Convert Electricity” and “Stop” functional states.

The relevant functional design tree is given in Figure 5.4. The following computer implementation of CD player is given in Figure 5.5 and Figure 5.6. The figures show the first level decomposition and the sub level of the functional state “Rotate CD”.

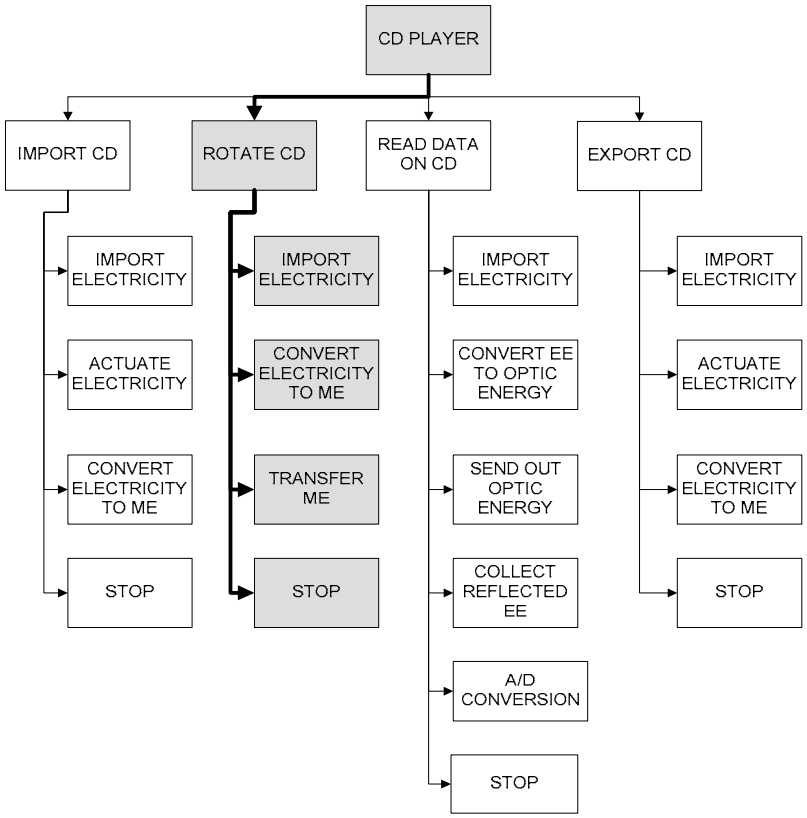


Figure 5.4 Functional Design Tree of CD player

Definition of Functional States

FIRST LEVEL:

$F(S) = \{ F_1, F_2, F_3, F_4, F_5 \}$

F1 = Import CD

F2 = Rotate CD

F3 = Read data on CD

F4 = Export CD / Stop

SECOND LEVEL

Functional decomposition of the functional state “Import CD” into its subfunctions follows:

$F(S) = \{ F_1, F_2, F_3, F_4 \}$

F1= import Electrical Energy

F2 = actuate electricity

F3 =convert electric energy to mechanical energy

F4= transfer mechanical energy

F5=Stop

Functional decomposition of the functional state “Rotate CD” into its subfunctions follows:

$F(S) = \{ F_1, F_2, F_3, F_4 \}$

F1= import Electricity

F2 =convert electric energy to mechanic energy

F3 =transfer mechanical energy

F4= Stop

Functional decomposition of the functional state “Read Data on CD” into its subfunctions follows:

$F(S) = \{ F_1, F_2, F_3, F_4 \}$

F1= import electricity

F2 =convert electric energy to optic energy

F3 =send out optic energy

F4= collect reflected energy

F5= analog to digital conversion

F6= Stop

Definition of Variables

FIRST LEVEL

p_{10} = availability of CD

p_{20} = Play command

SECOND LEVEL

The variables for the sub level of the functional state of "Rotate CD":

p_{10} = command

p_{20} = availability of CD

Instantiations of Variables

FIRST LEVEL

p_{11}=yes (there is CD inside)

p_{12}=no(there is no CD inside)

SECOND LEVEL

The instantiation variables for the sub-level of the functional state "Rotate CD":

p_{21}=yes (play the CD)
p_{22}=no(do not play the CD)

Decision Functions

FIRST LEVEL

DF={df_1 , df_2 , df_3 }
df_1= decision function for availability of CD
df_2= decision function for play command

SECOND LEVEL

The decision functions for the sub level of the function state "Rotate CD" follows:

DF={df_1, df_2}

df_1= decision function for command
df_2= decision function for availability of CD

DEFINITION OF I – MAPPINGS

FIRST LEVEL

I (availability, df_1)=1 I(play, df_2)=1
I(av. yes,F_2)=1 I(av. yes, F_3)=1 I(av. no, F_1)=1
I(pl. yes, F_1)=1 I(pl. yes, F_2)=1 I(pl. yes,F_3)=1
I(pl. no,F_4)=1

SECOND LEVEL

The I-Mappings for the sub level of the functional state “Rotate CD”

I(command, df_1)=1	I(availability, df_2)=1	
I(comm. yes, F_1)=1	I(comm. yes, F_2)=1	
I(comm. yes, F_3)=1	I(comm. no, F_4)=1	I(avail. yes, F_2)=1
I(avail. yes, F_3)=1	I(avail. no, F_4)=1	

DEFINITION OF O – MAPPINGS

FIRST LEVEL

O(df_1, av. yes)=1	O(df_2, play yes)=1	O(F_1, av. of CD)=1
O(df_1, av. no)=1	O(df_2,play no)=1	O(F_2, av. of CD)=1
O(F_3, av. of CD)=1	O(F_1, play com.)=1	O(F_2, play com.)=1
O(F_3, play com.)=1		

SECOND LEVEL

The O-Mappings for the sub level of the functional state “Rotate CD”

O(df_1,yes)=1	O(df_1, no)=1	O(df_2,yes)=1
O(df_2, no)=1	O(F_1,command)=1	O(F_2, command)=1
O(F_3, command)=1	O(F_1,av. of CD)=1	O(F_2, av. of CD)=1
O(F_3, av. of CD)=1		

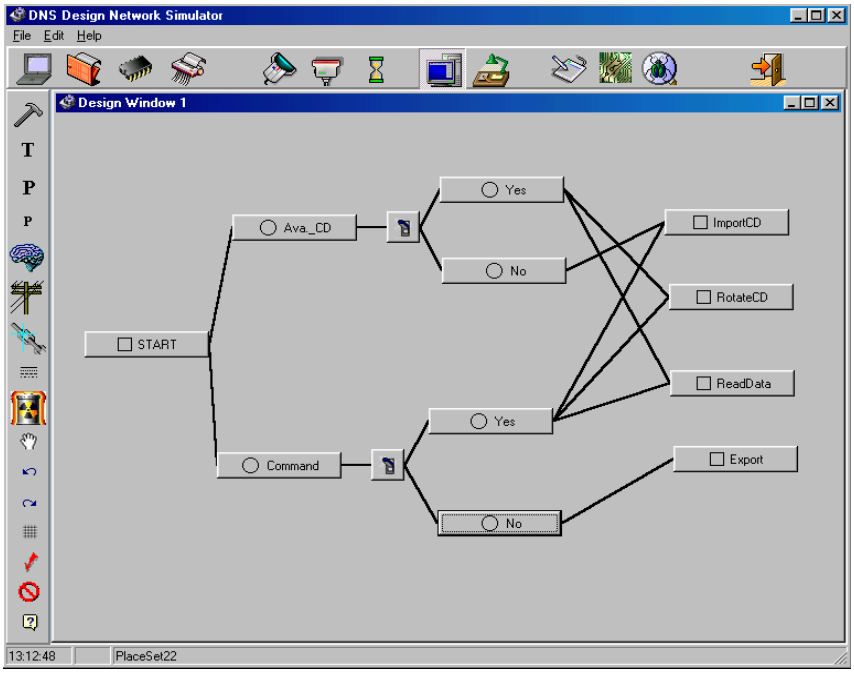


Figure 5.5 PNDN model of CD player

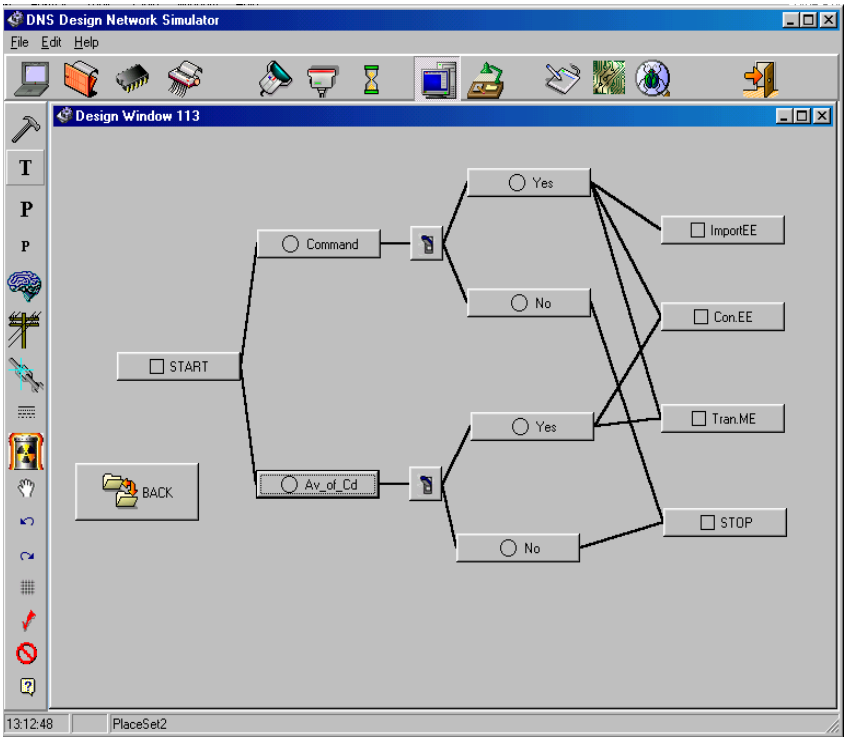


Figure 5.6 PNDN model of “Rotate CD” subfunction

5.3 N-PNDN Model of Coffee Machine

In the N-PNDN model of Coffee Machine “Start”, “Import Water”, “Import Coffee”, “Generate Heat”, “Mix Coffee and Water” come in the first decomposition level. The functional state “Generate Heat” is decomposed to “Import Electricity”, “Convert Electricity to Heat” and “Sense Heat” and “ Stop” functional states in the second level. In addition to that, the functional state in the first level “Mix Hot water and Coffee” is decomposed into “Guide Water” and “Generate Mixture” and “Stop” functional states. The relevant functional design tree is given in Figure 5.11. The following computer implementation of coffee machine is given in Figure 5.12 and Figure 5.13. The figures show the first level decomposition and the sub level of the functional state “Generate Heat”.

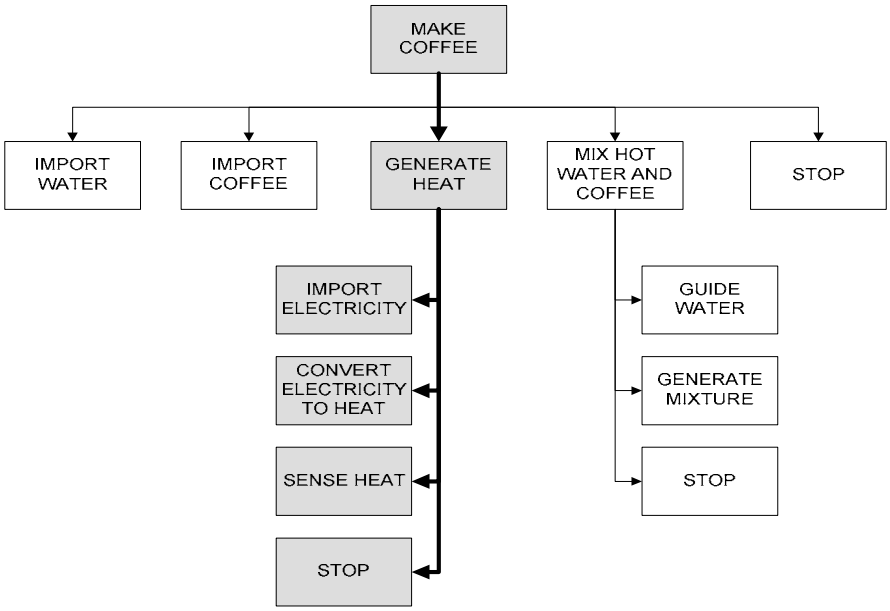


Figure 5.7 Functional Design Tree of Coffee Machine

Definition of Functional States

FIRST LEVEL:

$F(S) = \{ F_1, F_2, F_3, F_4, F_5 \}$

F1 = Import Water

F2 = Import Coffee

F3 = Generate Heat

F4 = Mix Hot Water and Coffee

F5 = Stop

SECOND LEVEL

Functional decomposition of the functional state "Generate Heat" into its subfunctions follows:

$F(S) = \{ F_1, F_2, F_3, F_4 \}$

F1= Import Electricity

F2 =Convert Electricity to Heat

F3 =Stop

Functional decomposition of the state "Mix Hot Water and Coffee" in to its subfunctions follows:

$F(S) = \{ F_1, F_2, F_3, F_4 \}$

F1= Guide Water

F2 =Generate Mixture

F3 =Stop

Definition of Variables

FIRST LEVEL

p_{10} = Check Water (information about existence of water)

p_{20} = Check Coffee (information about coffee in the machine)

p_{30} = Electricity (is electricity actuated or not?)

SECOND LEVEL

The variables for the sub-level of the functional state “Generate Heat”:

p_{10} =command (on or off)
p_{20} = Thermostat (is the water temperature is less or greater than 70)

The variables for the sub-level of the functional state “Mix Hot Water and Coffee”:

p_{10} =water flow (yes or no)
p_{20} = Filter (yes or no)

Instantiations of Variables

FIRST LEVEL

p_{11} = yes (there is water)
p_{12} = no (there is no water)
p_{21} = yes (there is coffee)
p_{22} = no (there is no coffee)
p_{31} = yes (electricity is actuated)
p_{32} = no (electricity is not actuated)

SECOND LEVEL

The instantiation variables for the sub-level of the functional state “Generate Heat”:

p_{11} = on

p_{12} = off

p_{21} = T<70 (temperature is below 70 C)

p_{22} = T>70 (temperature above70 C)

The instantiation variables for the sub-level of the functional state “Mix Hot Water and Coffee”:

p_{11} = yes-there is water flow)

p_{12} = no- there is no water flow

p_{21} = yes-coffee in filter

p_{22} = no-no coffee in filter

Decision Functions

FIRST LEVEL

DF={df_1 , df_2, df_3 }

df_1 = decision function for the water

df_2 = decision function for the coffee

df_3 = decision function for the electricity

SECOND LEVEL

Decision functions for the sub level of the functional state “Generate Heat”

DF={df_1, df_2 }

df_1= decision function for the water

df_2= decision function for the filter

Decision functions for the functional state “Mix Hot Water and Coffee”

DF={df_1, df_2 }

df_1= decision function for the command

df_2= decision function for the thermostat

DEFINITION OF I – MAPPINGS

FIRST LEVEL

I (check water, df_1) = 1 I(check coffee, df_2) =1 I(electricity, df_3) =1

I(yes, F_3) = 1 I (no, F_1) = 1 I(yes, F_3) = 1

I(no, F_2) = 1 I (yes, F_3) = 1 I (yes, F_4) = 1

I (no, F_5) = 1

SECOND LEVEL

The I-Mappings for the sub level of the functional state “Generate Heat”

I(on, F_1) =1 I(on F_2) =1

I(off, F_3) =1 I (T<70, F_1) = 1 I (T<70, F_2) = 1

I (T>70, F_2) = 1

The I-Mappings for the sub level of the functional state “Mix Hot Water aod Coffee”

I(yes, F_1) =1 I(yes F_2) =1

I(no, F_3) =1 I (yes, F_2) = 1 I (no, F_1) = 1

DEFINITION OF O – MAPPINGS

FIRST LEVEL

O (df_1,yes) = 1 O(df_2, yes) =1 O (df_3, yes) = 1

$O(df_1, no) = 1$ $O(df_2, no) = 1$ $O(df_3, no) = 1$

$O(F_1, check\ water) = 1$ $O(F_1, check\ coffee) = 1$

$O(F_1, electricity) = 1$ $O(F_1, check\ water) = 1$

$O(F_2, check\ coffee) = 1$ $O(F_2, electricity) = 1$

$O(F_3, check\ water) = 1$ $O(F_3, check\ coffee) = 1$

$O(F_3, electricity) = 1$

SECOND LEVEL

The O-Mappings of the sub level of functional state “Generate Heat”

$O(df_1, on) = 1$ $O(df_1, off) = 1$

$O(df_1, T>70) = 1$ $O(df_1, T<70) = 1$

$O(F_1, command) = 1$ $O(F_1, Thermostat) = 1$

$O(F_2, command) = 1$ $O(F_2, Thermostat) = 1$

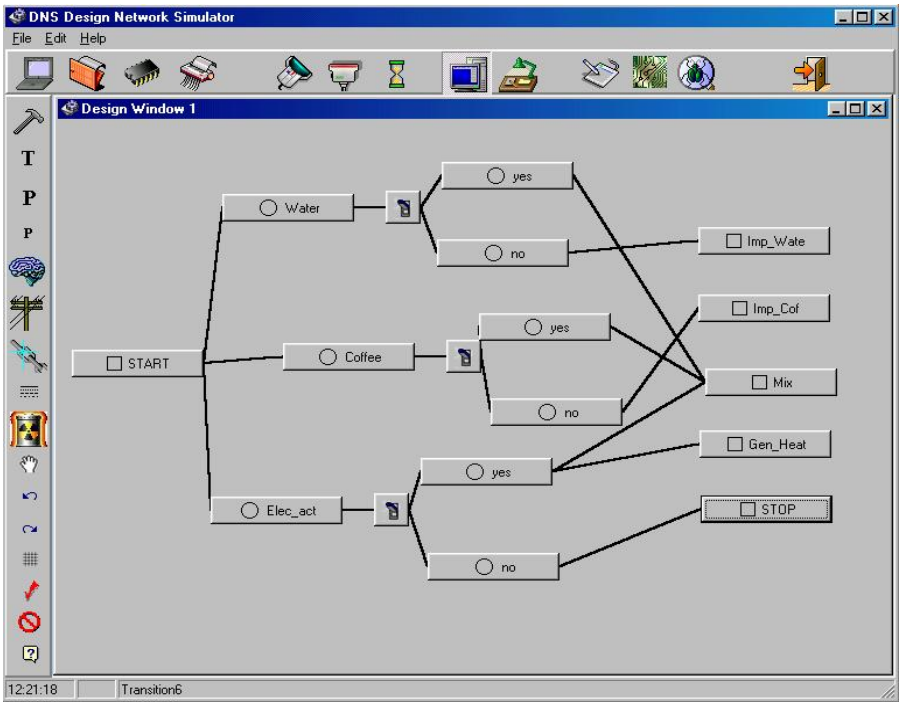


Figure 5.8 PNDN of. Coffee Machine

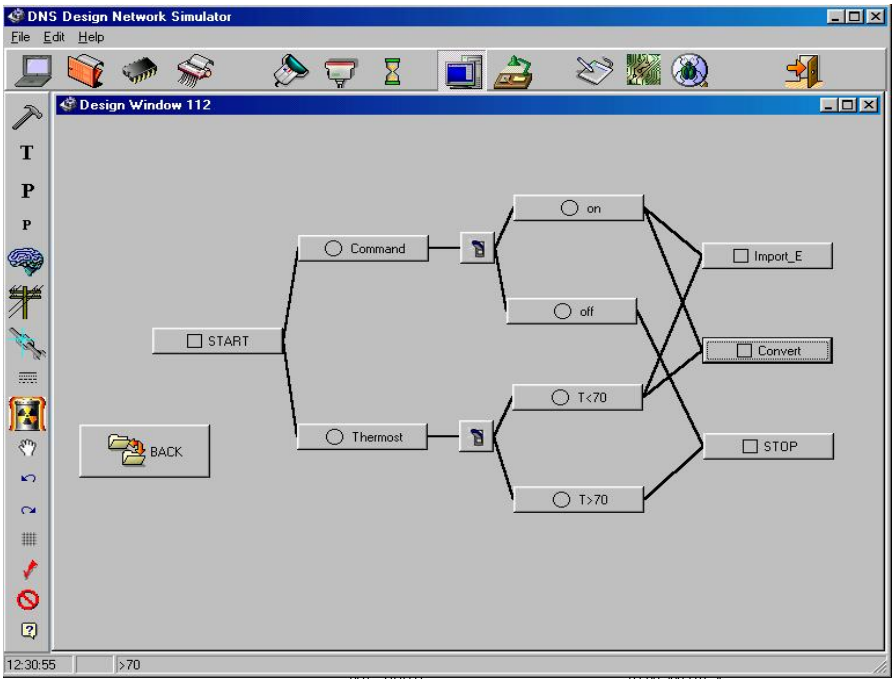


Figure 5.9 PNDN of Coffee Machine (continued)

5.4 N-PNDN Model of Lathe

In the N-PNDN model of Lathe (Korkmazel, 2001) “Start”, “Rotate Workpiece”, “Change Rotational Speed”, “Position the Tool”, “Feed The Tool” come in the first decomposition level. The functional state “Rotate Workpiece” is decomposed to “Rotate Counter Wise”, “Rotate Counter Clock Wise” and “Stop” functional states in the second level. In addition to that, the functional state in the first level “Change Rotational Speed” is decomposed into “Increase”, “Decrease” and “Stop” functional states. The functional state “Position the tool”, “Feed the Tool” is decomposed into “X-Dir”, “Y-Dir” and “Stop” functional states. The relevant functional design tree is given in Figure 5.11. The following computer implementation of coffee machine is given in Figure 5.12 and Figure 5.13. The figures show the first level decomposition and the sub level of the functional state “Rotate Work Piece”.

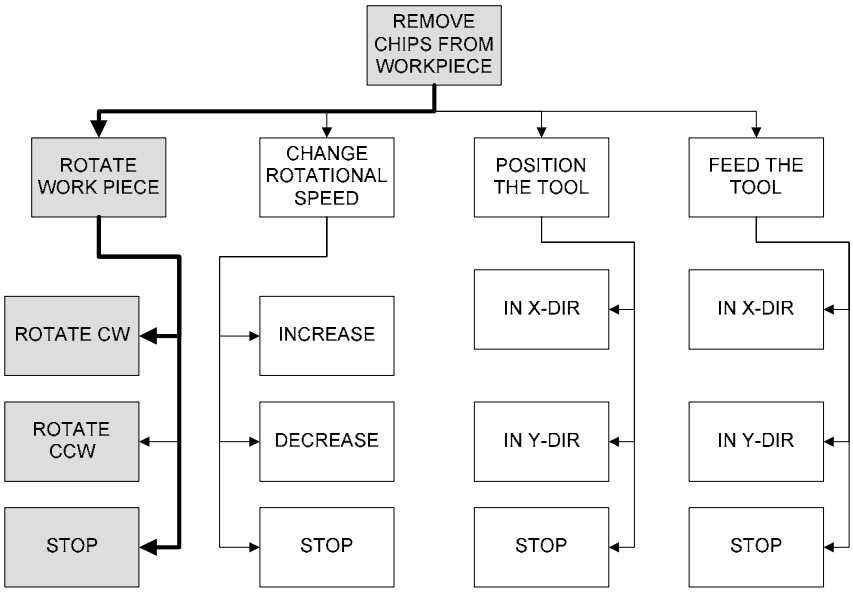


Figure 5.10 Functional Design Tree Lathe

Definition of Functional States

FIRST LEVEL:

F (S) = { F_1 , F_2 , F_3 , F_4, F_5 }

F1 = Rotate Work piece

F2 = Change Rotational Speed

F3 = Position the tool

F4 = Feed the tool

F5 = Stop

SECOND LEVEL

Functional decomposition of the functional state “Rotate Work Piece” into its subfunctions follows:

$F(S) = \{F_1, F_2, F_3, F_4\}$

F1= Rotate Work piece

F2 =Rotate CW

F3 =Rotate CCW

F4= Stop

Functional decomposition of the functional state “Change Rotational Speed” into its subfunctions follows:

$F(S) = \{F_1, F_2, F_3, F_4\}$

F1= Change Speed

F2 =Increase speed

F3 =Decrease speed

F4= Stop

Functional decomposition of the functional state “Position the Tool” into its subfunctions follows:

$F(S) = \{F_1, F_2, F_3, F_4\}$

F1= Position tool

F2 =Move toolpost in X-dir

F3 =Move toolpost in Y-dir

F4= Stop

Definition of Variables

FIRST LEVEL

p_{10} = Rotation

p_{20} = Speed

p_{30} = Position

SECOND LEVEL

The variables for the sub level of the functional state of "Rotate Work Piece":

p_{10} = Rotate
p_{20} = Direct

The variables for the sub level of the functional state of "Change Speed":

p_{10} = Sp.
p_{20} = Sit

The variables for the sub level of the functional state of "Position Tool":

p_{10} = pos
p_{20} = xpos
p_{30} = ypos

Instantiations of Variables

FIRST LEVEL

p_{11}: rot=0
p_{12}: rot =1
p_{21}: sp=0
p_{22}: sp=1
p_{31}: rot=0
p_{32}: rot=1

SECOND LEVEL

The instantiation variables for the sub-level of the functional state “Rotate Work piece”:

p_{11}: rot=0
p_{12}: rot =1
p_{21}: dir=0
p_{22}: dir=1

The instantiation variables for the sub-level of the functional state “Change Speed”:

p_{11}: SP=0
p_{12}: SP =1
p_{21}: increase
p_{22}: decrease

The instantiation variables for the sub-level of the functional state “Position Tool”:

p_{11}: pos=0
p_{12}: pos=1
p_{21}: xpos=0
p_{22}: xpos=1
p_{31}: ypos=0
p_{32}: ypos=1

Decision Functions

FIRST LEVEL

$DF=\{df_1 , df_2 , df_3 \}$

df_1= decision function for rotation
df_2= decision function for speed
df_2= decision function for position

SECOND LEVEL

The decision functions for the sub level of the function state “Rotate Work piece” follows:

$DF=\{df_1, df_2\}$

df_1= decision function for rotation
df_2= decision function for direction

The decision functions for the sub level of the function state “Change Speed” follows:

$DF=\{df_1, df_2\}$

df_1= decision function for speed increment
df_2= decision function for speed reduction

The decision functions for the sub level of the function state “Position the tool” follows:

$DF=\{df_1, df_2, df_3\}$

df_1= decision function for position

df_2= decision function for x position

df_3= decision function for y position

DEFINITION OF I – MAPPINGS

FIRST LEVEL

I (Rotation, df_1)=1	I(Speed, df_2)=1	I(Position, df_3)=1
I(rot=0,F_6)=1	I(rot, F_2)=1	I(rot=1, F_3)=1
I(rot=1, F_5)=1	I(speed=0, F_5)=1	I(speed=1,F_3)=1
I(pos=0,F_5)=1	I(pos=1,F_4)=1	

SECOND LEVEL

The I-Mappings for the sub level of the functional state “Rotate Work piece”

I(Rotation, df_1)=1	I(Direct, df_2)=1	
I(Rot=0, F_2)=1	I(Rot=1, F_3)=1	
I(Rot=0, F_4)=1	I(Dir=0, F_3)=1	I(Dir=1, F_2)=1

The I-Mappings for the sub level of the functional state “Change Rotational Speed”

I(SP, df_1)=1	I(Sit, df_2)=1	
I(SP=1, F_2)=1	I(SP=1, F_3)=1	
I(SP=0, F_4)=1	I(inc, F_2)=1	I(dec, F_3)=1

The I-Mappings for the sub level of the functional state “Position Tool”

I(pos, df_1)=1	I(xpos, df_1)=1	I(ypos, df_3)=1
I(pos=1, F_2)=1	I(pos=1, F_3)=1	I(pos=0, F_4)=1

$I(xpos=0, F_4)=1$ $I(xpos=1, F_2)=1$ $I(ypos=0,F_4)=1$
$I(ypos=1, F_3)=1$

DEFINITION OF O – MAPPINGS

FIRST LEVEL

$O(df_1,rot=0)=1$ $O(df_2,speed)=1$ $O(F_3,Rotation)=1$
$O(df_1,rot=1)=1$ $O(df_2,speed)=1$ $O(F_3,Speed)=1$
$O(F_1,Rotation)=1$ $O(F_2,Rotation)=1$ $O(F_4,Rotation)=1$
$O(F_1,Speed)=1$ $O(F_2,Speed)=1$ $O(F_4,Speed)=1$
$O(F_5,Rotation)=1$ $O(F_5,Speed)=1$

SECOND LEVEL

The O-Mappings for the sub level of the functional state “Rotate Work piece”

$O(df_1,rot=0)=1$ $O(df_2,Dir=0)=1$ $O(df_1,Rot=1)=1$
$O(df_2,Dir=1)=1$ $O(F_1,Rot)=1$ $O(F_2,Rot)=1$
$(F_1,Direct)=1$ $O(F_2,Speed)=1$ $O(F_3,Rot)=1$
$O(F_3,Direct)=1$

The O-Mappings for the sub level of the functional state “Change Speed”

$O(df_1,SP)=1$ $O(df_2,inc)=1$ $O(df_1,SP=1)=1$
$O(df_2,dec)=1$ $O(F_1,SP)=1$ $O(F_2,SP)=1$
$(F_1,Sit)=1$ $O(F_2,Sit)=1$ $O(F_3,SP)=1$
$O(F_3,Sit)=1$

The O-Mappings for the sub level of the functional state “Position Tool”

$O(df_1,pos=1)=1$	$O(df_2, xpos=1)=1$	$O(df_3,ypos=1)=1$
$O(df_1,pos=0)=1$	$O(df_2, xpos=0)=1$	$O(df_3, ypos=0)=1$
$O(F_1,pos)=1$	$O(F_2,pos)=1$	$O(F_3,pos)=1$
$O(F_1,xpos)=1$	$O(F_2,xpos)=1$	$O(F_3,xpos)=1$
$O(F_1,ypos)=1$	$O(F_3,ypos)=1$	$O(F_3,ypos)=1$

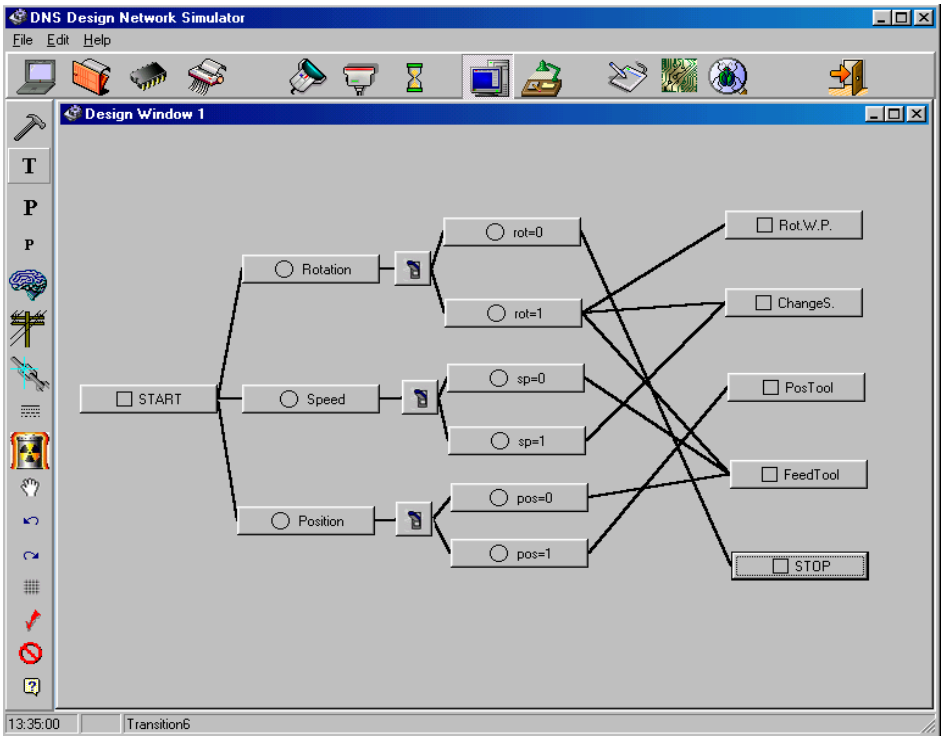


Figure 5.11 PNDN model for Lathe

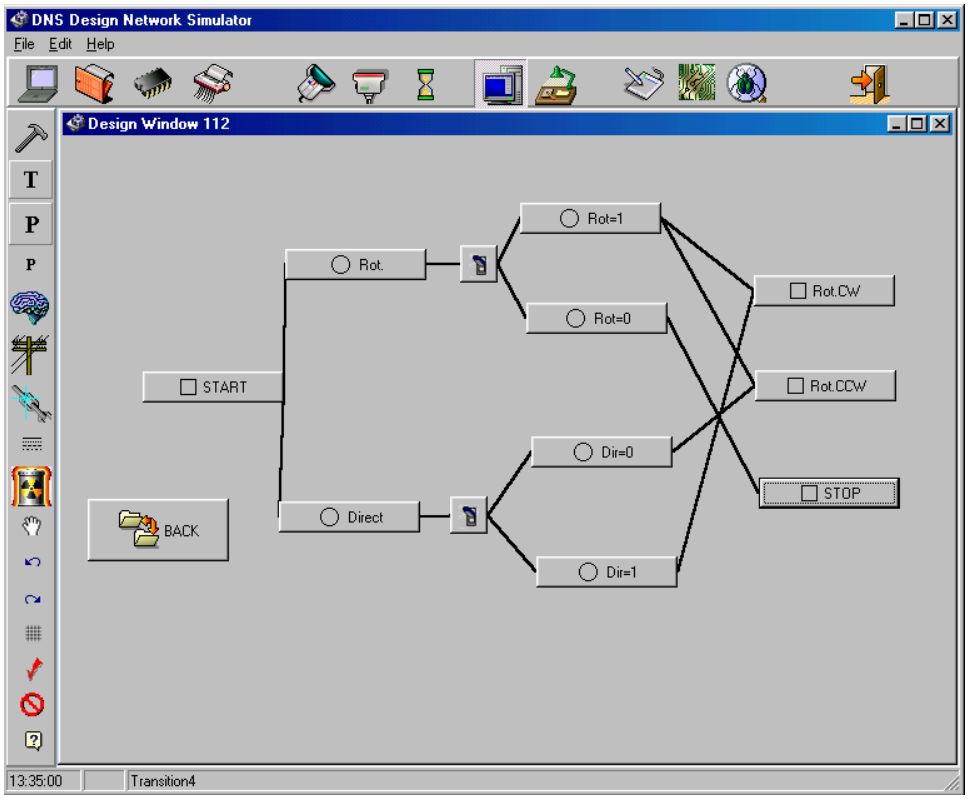


Figure 5.12 PNDN model of "Rotate Workpiece" subfunction

5.5 Evaluation of Case Studies

In this thesis with the computer implementation of N-PNDN, mechatronic systems are modeled as a network of embedded modules. In application N-PNDN brings the PNDN modules together with sub modules and therefore the functions of the system are modeled in every resolution level. While developing the algorithms for N-PNDN no modification is made on the original PNDN formalism other than the replacement of START transition with the sub PNDN modules which reveals the N-PNDN formalism. As it is expected PNDN modules in N-PNDN possessed the following properties:

1. Reachability: As a definition reachability is the set of all possible markings that can be reachable from an initial marking, M_0^V, in a PNDN therefore in N-PNDN. This property is used to check if the

all functional states are reachable from an initial marking to provide the use of related functional states. The PNDN modules in N-PNDN have also reachability property.

2. Concurrency: Concurrency is accomplished in the network through the development of N-PNDN as suggested by (Erden, 1999). In N-PNDN, firing transition can denote a child PNDN which means the machine has another state for the relevant subfunction. This has been shown in different case studies.
3. Liveness: PNDN and therefore N-PNDN is said to be live if one functional state fires under any reachable instantiation marking M^I. Liveness of N-PNDN guarantees dead-lock free operations in which two or more transitions are not allowed to fire simultaneously. This property is accomplished and supported by the program in all level of decomposition.

CHAPTER 6

CONCLUSIONS

N-PNDN is a design artifact modeling tool which models the mechatronic products at the multi level of functionality and is based on the PNDN theory. The architecture of N-PNDN is developed at the functional level since the computerized conceptual design tools need formal representation of functions and their relationships. This functional representation enables designers to find the best conceptual design solution in a more efficient way and in shorter time independent of any specific domain.

In general functionality of the system follows the functional decomposition hierarchy and subfunctions are related with the overall function. The theory of PNDN and therefore N-PNDN makes it possible to integrate local inferences and it models this integration. For this reason it is a suitable tool for modeling mechatronic products which consist of multidisciplinary engineering aspects.

The network structure makes it possible to define system subfunctions and also enables to model information, material and energy flows. In this study, PNDN formalism is conserved and this structure is extended in a network structure by replacing the upper level transitions with their sub PNDN modules. As a result N-PNDN is created and implemented. N-PNDN is a concurrent design modeling tool for the mechatronic products, and this criterion is considered while developing the algorithm. By definition concurrency is defined as a machine accomplishing more than one of its subfunctions simultaneously. With the introduction of N-PNDN multi resolution of mechatronic products together with concurrency is

incorporated in PNDN (Erden, 1999). This concludes that N-PNDN provides a concurrent structure since the systems consists of upper and lower level states.

In case studies functional decomposition hierarchy which is called Functional Design Tree in different resolution levels are investigated and applied to various mechatronic products. It has been realized that an important condition has to be satisfied that the means of accomplishing function should be specified in order to decompose the functions into subfunctions. This law was proposed as "Law of Vertical Causality" (Hubka 1976, Andreasen 1980) which states that the decomposition of a particular function into subfunctions is possible when a mean has been chosen to realize the function.

Another reason for the functional decomposition is to provide the modularity of the system since functional decomposition hierarchy is composed of function-subfunction structures at each abstract level. Therefore this structure can be considered as a functional module in functional design tree. Thus PNDN (also N-PNDN) has a modular structure and function subfunction modules can be represented in every resolution of FDT by this powerful tool.

N-PNDN's modularity feature enables us to model the mechatronic modules which consist of sensor, processing unit and an actuator. Those modules can find their physical representations in N-PNDN structure. These representations are places, decision functions, transitions which refer to sensor, processing, actuator respectively. Information flow is modeled in a structure that begins from upper level PNDN to lower level PNDN and no information flow is allowed in the same layer.

In this study, computer implementation of N-PNDN theory, the further development of DNS , is achieved successfully. The goals which had been specified in Chapter 1 and scope of the research part were:

1. Developing a new algorithm for the multi level PNDN.
2. Developing a new algorithm which enables the token flow between layers.
3. Transition from one level to another without loosing the previous design window.
4. User friendly graphical user interface
5. Exploring the modularity features of PNDN

At the end of this thesis the achieved targets which have been mentioned above:

1. Developing an algorithm for the construction of multi-level PNDN structure.
2. Constructing the N-PNDN modules separately and for every resolution level independent of each other which enables to model the complex systems and therefore high resolution levels can easily be handled.
3. Developing algorithm for the token flow in the multi level resolution which represents the information flow between the levels.
4. Developing an algorithm for the transition from one level to another without loosing the related data which belongs to the previous design or level.
5. Providing a user friendly GUI so that the people coming from different engineering disciplines can easily use the software package.
6. Supporting and manipulating of previously created designs.

Through the guidance of those achievements, the main contributions of this thesis work are:

1. It has been showed that PNDN is a modeling module for the mechatronic products and modules.
2. N-PNDN structure is applied and modeled to different products.
3. Modularity features of PNDN are explored.
4. Functional decomposition is used in N-PNDN
5. Multi resolutional features are incorporated in the program.
6. Concurrency is integrated and implemented by N-PNDN

Summarizing the advantages of N-PNDN yields:

1. N-PNDN allows us to create and model mechatronic modules which have different engineering components where these modules can find their interpretations.
2. N-PNDN facilitates the creation and modeling of mechatronic modules.
3. N-PNDN shortens the design time and gives the designer to evaluate the different design alternatives.
4. N-PNDN is a tool for visualization and simulation for the dynamic behavior of the system.
5. N-PNDN brings modularity which provides transparency, modifiability and incrementability.

The further developed DNS software has tool bar and control panel which provides the user to place the transitions, place sets, decision functions, in other words enables the user to access the tools that are used in network creation, token game and displaying the information dialog boxes for the network elements. It has also warning features for the users by text messages. In addition to that the tool tip feature which is provided by

Borland C++ Builder is also added to the program. A detailed user manual is also available in the help part of the software.

The goal of rapid multi level network creation is achieved by further developed DNS. The further developed DNS has the same rules for all applications and those are the automatically created functional states, links, and decision functions. Therefore the user does not need to redefine the elements during the design phase and this makes it easy to handle the networks and this results in faster and more efficient network network creation.

Having built on those features, during the design of the DNS, the operating system called Windows compatibility has been also taken into account. Like the all windows programs, the DNS has also “Open”, “Save” and “Print” basic features. As mentioned before all created networks are saved as text file and it enables the user for the further modifications by any text editor.

One other factor in this study is the simulation feature. The token flow which represents the information flow through the all network resolution levels is achieved. Computer implementation of deterministic token flow for the N-PNDN is completed and the non-deterministic part is left for the future study. The N-PNDN theory is in progress and therefore the further developed DNS, which is the main study of this thesis, is not final version.

6.1 FUTURE WORK

- Implementation of non-deterministic part for N-PNDN
- Developing the analysis part
- Developing for the modules for the information and energy flow through network in both single and multi level

The uncertainties in N-PNDN can be handled by using non deterministic token flow. Therefore computer implementation of the non deterministic part is also needed.

Analysis part of Petri Net simulation is also an important feature which has to be implemented to the program. This consists of “performance analysis” and “structural analysis”. Performance type of analysis is valid only for timed Petri Nets and therefore it is not suitable for DNS since PNDN theory is independent of time. ON the other hand PNDN has some properties like liveness or reachability which are inherited from the Petri Net theory. These properties are important for the analysis part hence it would result in improvements to add those features.

In this thesis after the applications, some improvements are made but it also needs some more application in order to have a better performance and reliability. Finally after completing these new implementations on DNS, it will be a more powerful software package for the automation of conceptual design phase.

REFERENCES

1) Acar, M., *Mechatronics Education and Training, 1st International Workshop on Mechatronic Design and Production*, 15-19 November 1993, METU, Ankara, Turkey, 1993.

2) Aleixos, N., Company, P., Contero, M., *Integrated modeling with top-down approach in subsidiary industries,* Computers in Industry 53, pp 97-116, 2004.

3) "Alpha/Sim" (2004) [Online] Available : http://www.alphatech.com/secondary/techpro/alphasim/alphasim

4) Alur, R. et al., *The Algorithmic Analysis of Hybrid Systems, Proc.11th International Conf. On Analysis and Optimization of Discrete Event Systems, Lecture Notes in Control and Information Sciences 199*, Springer – Verlag, pp 331-351, 1994.

5) Amerongen, J., *Mechatronic Design,* Mechatronics 13, pp.1045-1066, 2003.

6) Andreadakis, S. K., *Analysis and Synthesis of Decision Making Organizations*, Ph.D. Thesis, MIT, USA, 1988.

7) Andreasen M. M., *Machine Design and development*, Ph.D. Dissertation in Danish, Lund University, Lund, 1980.

8) Blanchard B. S. and Fabrycky W. J., *Systems Engineering and Analysis*, 3rd Ed., Prentice Hall, NJ, USA, 1998.

9) Boardaman, J., *Systems Engineering,* Prentice Hall International Ltd.*, Cambridge*, UK., 1990..

10) Brink, S. R., *A Petri Net Design Simulation and Verification Tool*, M.Sc Thesis, Department of Computer Engineering College of Engineering Rochester Institute of Technology, Rochester, New York, 1996.

11) Buur, J., Mechatronics, *A Theoretical Approach to Mechatronics Design*, Ph.D Thesis, Technical University of Denmark, IK Publication 90.74A, Lyngby, Denmark, 1990.

12) Buur, J., *Mechatronics Design Methodology, NATO ASI Workshop on the The Advancements and Applications of Mechatronics Design in Textile Engineering*, 5-16 April 1992, Side, Turkey, 1992.

13) Calvert, C., Charlie Calvert's Borland C++ Builder, MacMillan Computer Publishing, USA, 1997.

14) Coeling, Erik, J. A. Theo, *Assessment of Mechatronic System Performance at an Early Design Stage,* IEEE/ASME Transactions on Mechatronics, The Netherlands, Vol. 7, No. 3, September, 2002.

15) Counsell, J.M., Porter, I. *Schemebuilder Mechatronics: Design Principles for Controller Design, in accepted for Bath Workshop on Power Transmission & Motion Contro*l, Bath, 1998.

16) Eckel, Bruce, *Thinking in C++*, Prentice Hall, Second Edition,USA, 1996.

17) Erden, Z., *A Petri Net Based Inference Network for Design Automation at Functional Level Applied to Mechatronic Systems*, Ph.D. Thesis, Mechanical Engineering. Dept. METU, Türkiye, 1999.

18) Erden Z., Erkmen A. M., Erden A., *Generation of Functional Cells for a Mechatronic Design Network*, Proceedings of the Third International Conference on Mechatronics and Machine Vision in Practice, Volume 1, pp 233-238, 1996.

19) Erden Z., Erkmen A. M., Erden A., *Structuring of the SMDMnet: A design Inference Network for Mechatronic Systems,* Proc. The 2nd Int. Symposium on Intelligent Manufacturing Systems. 1998.

20) Erden Z., Erkmen A. and Erden A., *Automation of Conceptual Design: An implementation of Petri – Net on Mechatronic Design(I)*, Endüstri & Otomasyon, Vol.13, pp. 16-19, 1998.

21) Erden Z., Erkmen A. and Erden A., *Automation of Conceptual Design: An implementation of Petri – Net on Mechatronic Design(II), Endüstri & Otomasyon,* Vol.14, pp. 16-20, 1998.

22) Erden Z., Erkmen A. and Erden A, *A Petri Net Based Design Network with Applications to Mechatronic Systems*, SDPS Transactions: Journal of Integrated Design and Process Science, Vol.2, No:3, pp. 32-48, 1998.

23) Fraser, C. and Milne J., *Integrated Electrical and Electronic Engineering for Mechanical Engineers,* McGraw-Hill International Limited, Cambridge, UK, 1994.

24) Güroğlu S., *Implementation of an Algorithm for a Petri Net Based Design Inference Network*, M.S. Thesis, Mechanical Engineering Department, METU, Türkiye, 1999.

25) Henziger, T. A., Kopke, P. W., Puri, A. and Varaiya, P., *What is Decidable About Hybrid Automata*, Proc. 27th Annual Symposium on The Theory of computing ACM Press, pp 373-382, 1995.

26) Hopcroft, J. E. and Ullman, J. D., *Introduction to Automata Theory, Languages and Computation*, Addison Wesley Publ. Co. Inc.,1979.

27) Hubka V., *Theorie der Konstruktionprozesse*, Springer Verlag, Berlin, 1976.

28) Hubka V., Andreasen M. M. and Eder W. E., Practical Studies in Systmatic Design, Butterworth & Co Publishers, UK, 1988.

29) Iserman, R. *Modeling and Design Methodology for Mechatronic Systems.* IEEE/ASME Trans. on Mechatronics, vol. 1, no. 1, pp. 16-28, 1996.

30) Kannapan S. M., Marshek K. M., *Design Synthetic Reasoning: A Methodology for Mechanical Design*, *Research in Engineering Design*, Volume2, pp.221-238, 1991.

31) Kohavi, Z., *Switiching and Finite Automata Theory*, TATA, McGraw- Hill Publishing Company Ltd., India, 1978.

32) Korkmazel M., *Development of Multi Level Petri Net Based Design Inference Network*, M.S. Thesis, Mechanical Engineering Department, METU, Türkiye, 2001

33) Kusiak A., Szczerbicki E., Park K., A Novel Approach to Decomposition of Design Specifications and Search for Solutions, International Journal on Production Research, Volume 29, No. 7, pp. 1391-1406, 1991

34) Lewis, H. R. and Papadimitriu, C. H., *Elements of Theory of Computation*, Prentice Hall Inc., Australia, 1981.

35) Mortazavian, H. and Lin, F., *Foundations of a Logical Theory of Modeling and Control of Discrete Event Systems*, Proc. IFAC Int. Symposium on Distributed Intelligent Systems, DIS' 91, Arlington, VA, USA, 1991.

36) Oh, V. K., et al., *A Generic Framework for the Description of Components in the Design and Simulation of Mechatronic Products*, Proceedings of the Joint Hungarian-British International Mechatronics Conference, 21-23 September 1994, Budapest, Hungary, pp.515-520, 1994.

37) Pahl G., and Beitz W., *Engineering Design-A Systematic Approach,* The Design Council, London, UK, 1988.

38) "Petri Net Tools Database Quick Overview" (2004) [Online] Available :
http://www.daimi.au.dk/PetriNets/tools/quick.html, 2004

39) Porter, I. *Schemebuilder Mechatronic*s, Accepted for Engineering Design Conference, Brunel University, 1998.

40) Porter, I. *Schemebuilder Mechatronics*, Engineering Design Center, Lancaster University, UK, 2002.

41) Puri, A. and Varaiya, P.*, Decidability of Hybrid Systems with Rectangular Differential Inclusions*, Proc. 6th Workshop on Computer Aided Verification, LNCS 818, Springer-Verlag, pp.95-104, 1994.

42) Puri, A. and Varaiya, P., *Verification of Hybrid Systems Using Abstractions*, Hybrid Systems Iii LNCS 999, Springer Verlag, 1995.

43) Quin, S., F., Harrison, R., West, A., A., Jordanov, I., R., Wright, D., K., *A framework of web-based conceptual design,* Computers in Industry, pp 153 – 164, 2003.

44) Ramadge, P. J. G. and Wonham, W. M.*, the Control of Discrete Event Systems*, Proc IEEE, Vol.77, No.1, pp.81-89, 1989.

45) Reisig, W., Petri *Nets: An Introduction*, Springer Verlag, Germany, 1985.

46) Reisig, W., *A Primer in Petri Net Design*, Springer Verlag, Germany, 1992.

47) Ringstad P., *A Comparison of Two Approaches for Functional Decomposition the Function/Means Tree and the Axiomatic Approach*, Proceedings of the International Conference on Engineering Design, Volume 2, pp. 57-64, 1997.

48) Rzevski, G., *On conceptual design of intelligent mechatronic systems,* Mechatronics 13, pp.1023-1044, 2003

49) Suh, N., *Axiomatic Design Theory for Systems,* Research in Engineering Design, Springer Verlag, London, pp 189-209, 1998.

50) Tabak, D. and Levis, A. H., *Petri Net Representation of Decision Models*, IEEE Transactions on Systems, Man, and Cybernetics, Part A, Vol. SMC-15, No.6, pp.812-818, 1985.

51) Tokuz, C. L. and Jones, J. R., *A Representation for Hybrid Machines-Bond Graphs, Proceedings of the* 6^{th} *International Machine and Design Production Conference*, 21-23 September 1994, Ankara, Turkey, pp.199-208, 1994.

52) Zeigler, B. P., *DEVS Representation of Dynamical Systems: Event-Based Intelligent Control*, Proc. IEEE, Vol.77, No.1, pp.72-80, 1989.

APPENDIX A

PETRI NET TOOL SURVEY

The existing tools, the Petri Net type that they support and the operating system that they are used in as environment are given below Table A.1 which is taken from

Table A.1 Existing Tools for the Petri Nets

Overview	**Features**		**Environments**
	PN Supported	***Components***	
ALPHA/Sim	High-level Petri Nets Petri Nets with Time	Graphical Editor Token Game Animation Fast Simulation Simple Performance Analysis	SunOS Solaris MS Windows NT
ARP	Place/Transition Nets Petri Nets with Time	Fast Simulation State Spaces Place Invariants Transition Invariants Structural Analysis Simple Performance Analysis	MS DOS
Artifex	Object-oriented PNs High-level Petri Nets Place/Transition	Graphical Editor Token Game Animation Fast Simulation Structural	Sun, SunOS HP, HP-UX Silicon Graphics, IRIX PC, Linux

	Nets Petri Nets with Time	Analysis Advanced Performance Analysis	PC, MS Windows NT PC, MS Windows 2000 PC, MS Windows XP
CoopnTools	High-level Petri Nets	Graphical Editor Fast Simulation Structural Analysis	Java
CPN-AMI	High-level Petri Nets Place/Transition Nets	Graphical Editor Fast Simulation State Spaces Place Invariants Transition Invariants Structural Analysis services for modular modeling	Sun Linux Macintosh
CPN Tools	High-level Petri Nets Petri Nets with Time	Graphical Editor Token Game Animation Fast Simulation State Spaces Simple Performance Analysis Interchange File Format	PC, MS Windows 2000 PC, MS Windows XP
Design/CPN	High-level Petri Nets Petri Nets with Time	Graphical Editor Token Game Animation Fast Simulation State Spaces Simple Performance Analysis Interchange File Format	Sun HP Silicon Graphics Linux

DPNSchematic	Place/Transition Nets D-extended PN	Graphical Editor Fast Simulation Schematic Tool	PC, MS Windows 98 PC, MS Windows NT PC, MS Windows 2000 PC, MS Windows XP
EDS Petri Net Tool	Place/Transition Nets Stochastic Petri Nets Petri Nets with Time Colored Nets	Graphical Editor Token Game Animation Fast Simulation Structural Analysis Simple Performance Analysis Interchange File Format Transition programming language	MS Windows
ExSpect	High-level Petri Nets Place/Transition Nets Stochastic Petri Nets Petri Nets with Time Hierarchy in modeling	Graphical Editor Token Game Animation Fast Simulation Simple Performance Analysis Advanced Performance Analysis Simulation engine available as COM component	MS Windows
F-net	Stochastic Petri Nets Petri Nets with Time	Graphical Editor Token Game Animation Fast Simulation State Spaces Place Invariants Transition Invariants	MS Windows OS/2

		Structural Analysis Simple Performance Analysis Advanced Performance Analysis	
GDToolkit	High-level Petri Nets Place/Transition Nets	Automatic layout	Sun Linux MS Windows
GreatSPN	High-level Petri Nets Stochastic Petri Nets Petri Nets with Time	Graphical Editor Token Game Animation Fast Simulation State Spaces Condensed State Spaces Place Invariants Transition Invariants Structural Analysis Advanced Performance Analysis	Sun Linux
HiQPN-Tool	High-level Petri NetsStochastic Petri Nets	Graphical Editor Token Game Animation State Spaces Place Invariants Transition Invariants Advanced Performance Analysis Interchange File Format	Sun
HPSim	Place/Transition Nets Stochastic Petri	Graphical Editor Token Game Animation	PC, MS Windows 95 PC, MS

	Nets Petri Nets with Time	Fast Simulation Simple Performance Analysis	Windows 98 PC, MS Windows NT PC, MS Windows 2000 PC, MS Windows XP
INA	High-level Petri Nets Place/Transition Nets Petri Nets with Time	State Spaces Condensed State Spaces Place Invariants Transition Invariants Net Reductions Structural Analysis Simple Performance Analysis Advanced Performance Analysis Interchange File Format CTL-based model checker	Sun Linux MS Windows
INCOME Process Designer	High-level Petri Nets Place/Transition Nets Petri Nets with Time	Graphical Editor Token Game Animation Fast Simulation	Sun, SunOS HP, HP-UX PC, Linux PC, MS Windows 2000 Java
JARP	Place/Transition Nets	Graphical Editor Token Game Animation State Spaces Interchange File Format	Java
JFern	Object-oriented PNs High-level Petri Nets	Graphical Editor Token Game Animation Fast Simulation	Java

	Place/Transition Nets Petri Nets with Time	State Spaces Simple Performance Analysis Interchange File Format	
Maria	High-level Petri Nets Place/Transition Nets Modular high-level nets Labeled state transition systems	Token Game Animation Fast Simulation State Spaces Modular state spaces LTL model checker with fairness assumptions Very high-level data types and operations	Sun, SunOS 5.7 and 5.8 (32-bit and 64-bit) Digital, UNIX 4.0 Silicon Graphics, IRIX 6.5 (32-bit and 64-bit) HP-UX 11.22 NetBSD, FreeBSD, OpenBSD PC, Linux PC, MS Windows 95 and later Apple, Mac OS X 10.1
Marigold	Place/Transition Nets Data flow constructs	Graphical Editor	Java
MISS-RdP	High-level Petri Nets Place/Transition Nets Stochastic Petri Nets Petri Nets with Time Hybrid model (continuous and discontinuous)	Graphical Editor Token Game Animation Fast Simulation Advanced Performance Analysis	Sun MS Windows
The Model-Checking Kit	High-level Petri Nets Place/Transition	State Spaces Condensed State Spaces	Sun Linux

	Nets	CTL-/LTL-Model-Checking, Deadlock-Checking, Reachability-Checking	
Nevod	Inhibitor Petri nets	Graphical Editor Token Game Animation Fast Simulation	MS DOS
Opera	Petri Nets with Time	Graphical Editor Token Game Animation Fast Simulation State Spaces Condensed State Spaces Place Invariants Transition Invariants Net Reductions Structural Analysis Simple Performance Analysis Advanced Performance Analysis Interchange File Format	MS DOS
PACE	High-level Petri Nets Place/Transition Nets Stochastic Petri Nets Petri Nets with Time Attributed Petri Nets	Graphical Editor Token Game Animation Fast Simulation Net Reductions Fuzzy Technics, Net Optimizations	PC, MS Windows 95 PC, MS Windows 98 PC, MS Windows NT PC, MS Windows 2000 PC, MS Windows XP

PED	Place/Transition Nets Petri Nets with Time	Graphical Editor	Sun Linux
PEP	High-level Petri Nets Place/Transition Nets Petri Nets with Time	Graphical Editor Token Game Animation State Spaces Condensed State Spaces Place Invariants Transition Invariants Net Reductions Structural Analysis Interchange File Format Model Checking Petri Net Generators	Sun Linux
Petrigen	Place/Transition Nets	Graphical Editor Token Game Animation synthesis	PC, Linux PC, MS Windows 95 PC, MS Windows 98 PC, MS Windows NT PC, MS Windows 2000 PC, MS Windows XP
Petri Net Kernel	High-level Petri Nets Place/Transition Nets DAWN-Nets User definable	Graphical Editor Token Game Animation Interchange File Format INA-pilot User definable	Java
Petri Net Toolbox	Place/Transition Nets Stochastic Petri Nets Petri Nets with	Graphical Editor Token Game Animation Fast Simulation State Spaces	

	Time GSPN	Place Invariants Transition Invariants Structural Analysis Simple Performance Analysis Advanced Performance Analysis Interchange File Format	
PetriSim	High-level Petri Nets Place/Transition Nets Petri Nets with Time	Graphical Editor Fast Simulation	MS DOS
Platform Independent Petri Net Editor	Place/Transition Nets	Graphical Editor Token Game Animation Fast Simulation State Spaces Condensed State Spaces Place Invariants Transition Invariants Structural Analysis Simple Performance Analysis Interchange File Format Extensible Analysis Modules and File Formats	Java
PNSim	Place/Transition Nets	Graphical Editor Token Game Animation	MS Windows Java

		Structural Analysis Simple Net Analysis	
PNtalk	High-level Petri Nets Petri Nets with Time	Graphical Editor Token Game Animation Fast Simulation Simple Performance Analysis	Sun MS Windows
Poses++	High-level Petri Nets	Fast Simulation	Sun Linux MS Windows
Predator	Place/Transition Nets Stochastic Petri Nets hierarchical Petri Nets (Subnets)	Graphical Editor Place Invariants Transition Invariants Interchange File Format Dynamic Loading of Analysis modules	Java
PROD	High-level Petri Nets Place/Transition Nets	State Spaces Condensed State Spaces LTL Model Checking, CTL Model Checking	Sun, SunOS HP, HP-UX PC, Linux PC, MS Windows 95 PC, MS Windows NT
Renew	Object-oriented PNs High-level Petri Nets Place/Transition Nets Petri Nets with Time	Graphical Editor Token Game Animation Fast Simulation Interchange File Format	Java
Romeo	Petri Nets with Time	Graphical Editor State Spaces	PC, Linux Macintosh, Mac OS X

SEA	High-level Petri Nets Petri Nets with Time	Graphical Editor Token Game Animation Fast Simulation Abstract Graphical Visualization	Sun
SIPN-Editor	 Interpreted Petri Nets	Graphical Editor Code Generation for PLC, Translation into SMV Code	Java
SimulaWorks	Place/Transition Nets Stochastic Petri Nets Petri Nets with Time	Graphical Editor Token Game Animation Fast Simulation	PC, MS Windows 95 PC, MS Windows 98 PC, MS Windows NT PC, MS Windows 2000 PC, MS Windows XP
SPNP	High-level Petri Nets Stochastic Petri Nets	Advanced Performance Analysis	PC, MS Windows 98
StpnPlay	Stochastic Petri Nets Petri Nets with Time	Graphical Editor Fast Simulation Simple Performance Analysis Interchange File Format	PC, MS Windows 95 PC, MS Windows 98 PC, MS Windows NT PC, MS Windows 2000 PC, MS Windows XP
SYROCO	High-level Petri Nets Petri Nets with Time	Graphical Editor Fast Simulation Simple Performance Analysis	C++

	Dynamic instantiation of nets, C++ code associated to transitions and places, priority	Interchange File Format C++ code generation	
TimeNET	High-level Petri Nets Place/Transition Nets Stochastic Petri Nets Petri Nets with Time	Graphical Editor Token Game Animation Fast Simulation State Spaces Place Invariants Structural Analysis Simple Performance Analysis Advanced Performance Analysis Interchange File Format	Sun Linux
Tina	Place/Transition Nets Petri Nets with Time Time Petri Nets	Graphical Editor State Spaces Condensed State Spaces Place Invariants Transition Invariants Structural Analysis State Class Spaces	Sun Linux MS Windows
Visual Object Net ++	Place/Transition Nets Petri Nets with Time Hybrid Dynamic Nets and Hybrid Object Nets	Graphical Editor Token Game Animation Fast Simulation Structural Analysis Simple Performance Analysis supports object	MS Windows

		hierarchies	
WebSPN	Stochastic Petri Nets	Graphical Editor Token Game Animation Advanced Performance Analysis prd, prs, pri memory policies	Java
WINSIM	Generalized Evaluation Nets Extended Petri nets	Fast Simulation	PC, MS Windows 98 PC, MS Windows NT PC, MS Windows 2000 PC,Windows XP

Table A.1 Existing Tools for the Petri Nets